Desarrollo de infraestructura GTD multiplataforma

GRADO EN INGENIERÍA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO

Andrés Herreros Manotas
Adrián Monteagudo Sampedro

Junio 2018

Director: Juan Carlos Sáez Alcaide

Desarrollo de infraestructura GTD multiplataforma

Memoria de Trabajo de Fin de Grado

Andrés Herreros Manotas
Adrián Monteagudo Sampedro

Director: Juan Carlos Sáez Alcaide

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid
Junio 2018

Resumen

La metodología GTD (Getting Things Done) diseñada por David Allen constituye actualmente uno de los mecanismos más eficientes de organización personal. Su objetivo es permitirnos lograr la máxima productividad mediante el almacenamiento de las tareas, proyectos y actividades a realizar en un lugar específico. En la actualidad existen múltiples aplicaciones para ayudarnos a poner en práctica la filosofía GTD. Algunas de estas aplicaciones permiten almacenar nuestra información GTD en la nube, para así acceder a ella cómodamente desde distintos dispositivos (p.ej., PC de sobremesa, tablet o móvil). Sin embargo, muchas de las aplicaciones disponibles son propietarias, y las empresas que ofrecen el servicio tienen acceso a nuestra información GTD personal, lo cual puede afectar muy negativamente a la privacidad de los usuarios.

En este proyecto se ha desarrollado una infraestructura GTD multiplataforma compuesta de un servidor, que permita almacenar la información en la nube, y de un conjunto de aplicaciones cliente para distintos dispositivos y sistemas operativos (Linux, Mac OS X, Android, etc.). Como servidor se plantea el uso y posible adaptación del desarrollado en el marco del proyecto de código abierto Tracks, que permite que los datos personales puedan almacenarse en una máquina privada gestionada por un conjunto reducido de usuarios o por una organización. Para simplificar el desarrollo de las aplicaciones cliente, se ha empleado la librería Kivy que permite construir aplicaciones gráficas multiplataforma en Python. Las aplicaciones cliente están provistas de un modo offline, que permite al usuario trabajar con el sistema GTD cuando no se dispone de conexión a Internet.

Palabras clave: GTD, Tracks, Kivy, REST API, Python, Proyecto, Contexto, Tarea

Abstract

The GTD methodology (Getting Things Done) designed by David Allen constitutes currently one of the most efficient mechanisms of personal organization. Its objetive is to allow us to achieve maximum productivity by storing the tasks, projects and activities to be carried out in a specific place. Currently there are multiple applications to help us implement the GTD philosophy. Some of these applications allow us to store our GTD information in the cloud, in order to access it conveniently from different devices (eg, desktop, tablet or mobile PC). However, many of the available applications are proprietary, and the companies that offer the service have access to our personal GTD information, which can negatively affect the privacy of users.

In this project has been developed a multiplatform GTD infrastructure composed of a server, which allows to store the information in the cloud, and a set of client applications for different devices and operating systems (Linux, Mac OS X, Android, etc.). As a server, we propose the use and possible adaptation of the one developed within the framework of the open source project Tracks, which allows personal data to be stored on a private machine managed by a reduced set of users or by an organization. To simplify the development of client applications, the Kivy library has been used to create multiplatform graphical applications in Python. The client's applications have an offline mode, which allows the user to work with the GTD system when there is no Internet connection.

Keywords: GTD, Tracks, Kivy, REST API, Python, Project, Context, Todo

Índice general

Agradecimientos

Queremos agradecer en primer lugar a nuestro director de proyecto, Juan Carlos Sáez Alcaide, por darnos la oportunidad de aprender y crecer con este proyecto que nos aporta grandes conocimientos para nuestro futuro, ayudándonos continuamente con nuestros problemas y haciendo posible que lo consigamos. Nos sentimos verdaderamente realizados de alcanzar esta meta tratando con tecnologías que hasta ahora no conocíamos, y que sin su ayuda no habría sido posible, pero dándonos la libertad de decisión para que ahora podamos utilizar lo aprendido sin su ayuda.

También queremos mencionar a nuestros familiares, amigos y compañeros que nos han acompañado durante este tiempo. Sin ellos tampoco hubiese sido posible llevar a cabo todo el trabajo que nos ha exigido este proyecto.

Capítulo 1

Introducción

GTD son las siglas de Getting Things Done, término creado por David Allen y que podemos traducir como "Organízate con eficacia", ya que así se hizo la traducción a español de su libro recogido en la Bibliografía (ver cita [1]). GTD es una metodología que ayuda a una persona a gestionar las tareas que lleva a cabo durante el día, permitiendo que se centre en las tareas actuales y no tenga que estar recordando o pendiente de las que están por venir.

GTD define un flujo de trabajo con 5 etapas que nos hacen desarrollar nuestro trabajo de manera eficiente: **Recopilar**, **Procesar**, **Organizar**, **Evaluar** y **Hacer**. En la sección 2.1 se explica el concepto con más detalle.

A continuación explicaremos la motivación para realizar este proyecto.

1.1. Motivación

Garantizar la privacidad de los usuarios de herramientas GTD es una de las principales motivaciones de este trabajo. Está claro que hoy en día son innumerables las aplicaciones que existen para que organicemos nuestro día a día (Things [13], Chandler [14], etc.). Lamentablemente, la mayor parte de ellas tienen un problema potencial: la recopilación de datos que posteriormente venden a terceros.

Por ello, planteamos la idea de tener control sobre nuestros datos en un servidor privado. Y, ¿por qué un servidor y no únicamente en nuestro dispositivo? Gracias al servidor podremos tener una copia de seguridad en la nube, además de tener disponibles nuestros datos en múltiples dispositivos.

1.2. Objetivos

Como hemos comentado anteriormente, son innumerables las aplicaciones que existen hoy en día en el mercado para organizar tu día a día pero no garantizan que nuestros datos estén a salvo. Además, algunas no ofrecen soporte offline o la posibilidad de usarla en todos nuestros dispositivos. Vamos a realizar una introducción sobre tres características de nuestra infraestructura, que se ampliarán en el capítulo 4.

1.2.1. Tracks como opción

En este proyecto proponemos el uso de Tracks (ver sección 4.1.2) como servidor de nuestra infraestructura GTD. Tracks está implementada con Ruby on Rails, posee un servidor web (WEBrick, aunque nosotros lo hemos realizado con Apache) que puede ser ejecutado en cualquier equipo/servidor en el que se pueda instalar Ruby. A pesar de que Tracks no posee clientes específicos (sólo interfaz web) para distintas plataformas, ésta ofrece un API REST que permite la interacción con el servidor desde una aplicación cliente construida específicamente.

1.2.2. Infraestructura multiplataforma

Nuestra infraestructura se compone de dos componentes, ambos multiplataforma: servidor(Tracks), y cliente (kvtracks). Como hemos comentado, Tracks se puede instalar en cualquier equipo en el que esté Ruby disponible. El cliente, kvtracks, está implementado en Python y, junto con el framework Kivy (ver en sección 3.2.3) conseguimos una aplicación que se puede ejecutar en diferentes sistemas operativos como Windows, Mac, iOS, Android, etc.

1.2.3. Modo offline

Una de las funcionalidades más útiles en nuestra aplicación es la posibilidad de usarla sin necesidad de conexión. Los datos se almacenan en una base de datos ubicada en el dispositivo donde se ejecuta el cliente y gestionada automáticamente por la aplicación kvtracks, hasta que el dispositivo vuelva a tener conexión y se guarden en la nube (servidor Tracks). La base de datos tiene un gestor llamado SQLite (ver sección 4.2.2).

1.3. Plan de trabajo

Para alcanzar los objetivos se realizaron las siguientes tareas:

1. Planificación del trabajo que realizaría cada uno de los miembros del equipo de trabajo del proyecto (recogido en el apéndice D).
2. Búsqueda y lectura de documentación acerca de los lenguajes de programación, tecnologías y herramientas a utilizar.
3. Instalación y despliegue de la aplicación web Tracks (servidor de la infraestructura) sobre una máquina virtual.
4. Instalación de las herramientas de desarrollo (Python, Kivy, etc.) para la implementación de kvtracks.
5. Diseño de los bocetos de kvtracks con balsamiq (ver apéndice C)
6. Desarrollo de prototipos de aplicaciones cliente con Python para familiarizarnos con las distintas tecnologías utilizadas.
7. Combinación de los prototipos creados anteriormente para comprobar la compatibilidad entre todas las tecnologías utilizadas.
8. Construcción del APK con Buildozer, previa lectura de documentación e instalación.
9. Análisis del funcionamiento interno de Tracks.
10. Modificación/Adaptación de los bocetos creados para el uso exclusivo de los componentes de la interfaz de usuario (widgets) soportados por Kivy
11. Implementación de los dos componentes de kvtracks: frontend y backend.
12. Integración de los dos componentes de kvtracks.
13. Pruebas de funcionamiento y corrección de errores.
14. Empaquetado y despliegue de la aplicación.

Cabe destacar que el desarrollo de algunas de las tareas se realizó en paralelo, y que en otras tareas cada miembro del equipo se ocupó de una parte específica, según lo acordado en la tarea 1.

1.4. Estructura de la memoria

El resto de la memoria se organiza de la siguiente forma:

- En el **capítulo 2** se proporciona una descripción detallada de la metodología GTD y se indica cómo puede ponerse en práctica con la aplicación web de Tracks.
- En el **capítulo 3** se explica el desarrollo de todas las tecnologías utilizadas para la implementación de la infraestructura: Python, Kivy, Buildozer y API REST.
- En el **capítulo 4** se resumen de los componentes software que componen el backend de la aplicación.
- En el **capítulo 5** se resumen de los componentes software que componen el frontend de la aplicación y los casos de uso.
- En el **capítulo 6** se muestran las conclusiones del proyecto.
- Además, se incluyen diversos **apéndices** en nuestra memoria. En el primero y en el segundo se localizan la introducción y las conclusiones en inglés. En el tercero se incluyen los bocetos del diseño de la interfaz, y en el cuarto el trabajo de cada miembro del equipo de trabajo de este TFG.

Capítulo 2

GTD y Tracks

A lo largo de este capítulo se explicará a fondo que es GTD y cómo se pone en práctica con la aplicación Tracks.

2.1. Filosofía GTD

Como comentamos en la introducción (ver capítulo 1), GTD es un método de organización de nuestras tareas diarias para maximizar nuestro rendimiento. Es un método aplicable tanto a la vida personal como a la profesional, que propone eliminar las preocupaciones acerca de lo que tenemos que hacer próximamente, pasándolas a un sistema externo que recuerde todo por nosotros y de esta manera centrarnos en lo que estamos haciendo actualmente. Allen creó esta metodología en busca de una solución a todas las tareas que pensamos y no se pueden realizar al momento, todas las responsabilidades de más que tenemos, la mezcla que se produce entre la vida profesional y la vida personal, o los sistemas que permiten una visión global de todo lo que tenemos que hacer, pero que no consiguen eliminar el agobio producido por la cantidad de trabajo.

Gestionando adecuadamente todas nuestras tareas el nivel de estrés se verá disminuido. Para realizar dichas tareas, Allen propone lo siguiente:

- Sacar de nuestra cabeza la tarea para ponerla en el sistema.
- Determinar qué hacer para llevarlo a cabo.
- Llevar a cabo revisiones para comprobar que no queda nada pendiente.

El núcleo de GTD reside en transformar todas aquellas cosas que nos preocupan en

acciones que podemos almacenar, y posteriormente realizar, y así finalizar nuestra preocupación. Allen define esas "cosas" como acciones/tareas que han entrado en nuestra mente, pero que no ocupan el lugar correcto y que crean dicha preocupación. El objetivo es por tanto transformar las "cosas" en acciones, cuyo proceso consta de 5 fases clave: **recopilar**, **procesar**, **organizar**, **evaluar** y **hacer**.

2.1.1. Recopilar

Como bien indica el nombre de la fase, consiste en recopilar todo aquello que nos preocupa y agruparlo. De esta forma ya hemos dado un paso, todo aquello que nos preocupa y teníamos disperso ahora lo tenemos unido y localizado.

Para la recolección serviría cualquier elemento en el que anotar aquello que nos inquieta: un cuaderno, un ordenador, un teléfono móvil, etc.

El orden de recolección que propone Allen es empezar por las preocupaciones de cosas físicas, como puede ser recoger una habitación, y posteriormente por las preocupaciones que tenemos en la cabeza.

2.1.2. Procesar

Tras recopilar todo aquello que nos preocupa es hora de procesarlo y convertirlo en soluciones o, como Allen define, convertirlo en acciones, las reglas que establece Allen son las siguientes:

- Ninguna preocupación es más importante que otra, TODO se procesa, desde lo primero que apuntamos hasta lo último.
- Procesar la preocupación una sola vez.
- Una vez que procesamos la preocupación y la hemos convertido en acción, no puede volver a la fase anterior, es decir, no puede volver a ser recopilada como una preocupación puesto que ya le hemos puesto solución.

Para llegar de preocupación a solución tenemos que hacernos una pregunta: ¿Queremos hacer algo con lo que nos preocupa y hemos recogido? Tenemos 2 respuestas muy simples: NO o SÍ:

No hacer nada:

Si no queremos hacer nada con la preocupación, tenemos tres vías de actuación:

- Eliminarlo de la lista.
- Si creemos que tiene interés como información pero no podemos convertirlo en acción, lo archivamos.
- Si tiene interés como información, lo podemos convertir en acción, pero no queremos hacerlo ahora lo pasamos la lista de hacer quizás más tarde.

Realizar algo con nuestra preocupación:

Si queremos hacer algo con la preocupación, tan solo tendremos que transformarla en una acción que podamos ejecutar. El problema viene cuando para acabar con el problema tenemos que realizar varias acciones. Es lo que Allen denomina *proyecto*. Para gestionar un proyecto debe ser añadido en la lista de proyectos y debemos visualizar el resultado final al que queremos llegar cuando todas las acciones hayan sido realizadas.

Los pasos a seguir en la gestión de un proyecto son:

1. Definir qué queremos conseguir.
2. Visionar el resultado: imaginar que hemos alcanzado la solución y de qué manera lo hemos hecho.
3. Tormenta de ideas: empezar a pensar ideas que nos permiten finalizar el proyecto.
4. Organizar las ideas: una vez que hemos terminado con la tormenta de ideas, comenzamos a establecer prioridades.
5. Determinar la próxima acción, el proceso por el cual convertimos una idea en una acción física.

Haciendo alusión al último punto de cómo gestionar un proyecto, tenemos que saber qué hacer con la próxima acción, para ello Allen propone tres opciones:

1. Si la acción lleva poco tiempo, hazla.
2. Si no está en nuestra mano hacerlo pero si en la de otra persona, delégala.
3. Si la acción no se tiene que hacer inmediatamente y hay un tiempo para hacerla,

difiérela.

Es aconsejable que todas las acciones tengan un *contexto* asignado. El contexto delimitará el lugar físico donde se realiza cada acción y nos ayudará a organizarnos mejor.

2.1.3. Organizar

Una vez que todo ha sido procesado es hora de organizarlo:

- Lo que no podamos hacer o no queramos hacer ahora acabará en la lista de quizás algún día o en la basura.
- Lo que sí podamos hacer y queramos hacerlo acabará en la lista de proyectos, lista de acciones, lista de próximas acciones, listas de espera, etc.

Según vayamos completando las acciones éstas pasaran a una nueva lista: *checklist*

2.1.4. Evaluar

La fase de evaluar consiste en asegurarse de que nada queda sin hacer. Las revisiones pueden ser diarias, semanales, mensuales, trimestrales, etc.

2.1.5. Hacer

La última fase implica determinar qué próximas acciones haremos en función del tiempo del que disponemos, de las que podamos hacer o de la prioridad que tenga cada una. Para ello tenemos que tener una visión realista de lo que sí y lo que no podemos hacer en función de la situación en la que nos encontremos.

2.2. Conceptos básicos

Tras la explicación de en qué consiste GTD y sus 5 fases, independientemente de cómo se lleva a cabo cada una, es importante recalcar el significado de tres conceptos que nos servirán para entender cómo funciona la aplicación Tracks (ver sección 2.3):

- **Contexto**: Lugar físico en el que se desarrollan las tareas, pertenezcan a un proyecto o sean individuales.

- **Proyecto**: Conjunto de tareas que tras su realización nos permitirán llegar a la solución (previamente ideada) de un problema/preocupación.
- **Tarea**: Acción perteneciente a un proyecto (o no) que se desarrolla dentro de un contexto y que su realización nos puede proporcionar directamente una solución (si no pertenece a un proyecto) o acercarnos un poco más a la solución.

Para entender todos los conceptos mucho mejor, imaginemos un profesor que imparte clase en la Universidad Complutense de Madrid. Principalmente su vida se desarrolla entre la facultad y su casa, por lo que siguiendo la metodología GTD dos contextos podrían ser: UNIVERSIDAD y CASA.

Si el profesor imparte dos asignaturas -por ejemplo base de datos y programación-, en cada asignatura tendrá que impartir clase, corregir ejercicios, idear prácticas, corregir prácticas, corregir exámenes... por lo que los proyectos podrían ser BBDD y PROGRAMACIÓN. Dentro de cada uno de los proyectos se encontrarán las tares mencionadas anteriormente y que pertenezcan a cada asignatura.

Un esquema de cómo sería la metodología según el esquema propuesto sería el reflejado en la figura 2.1

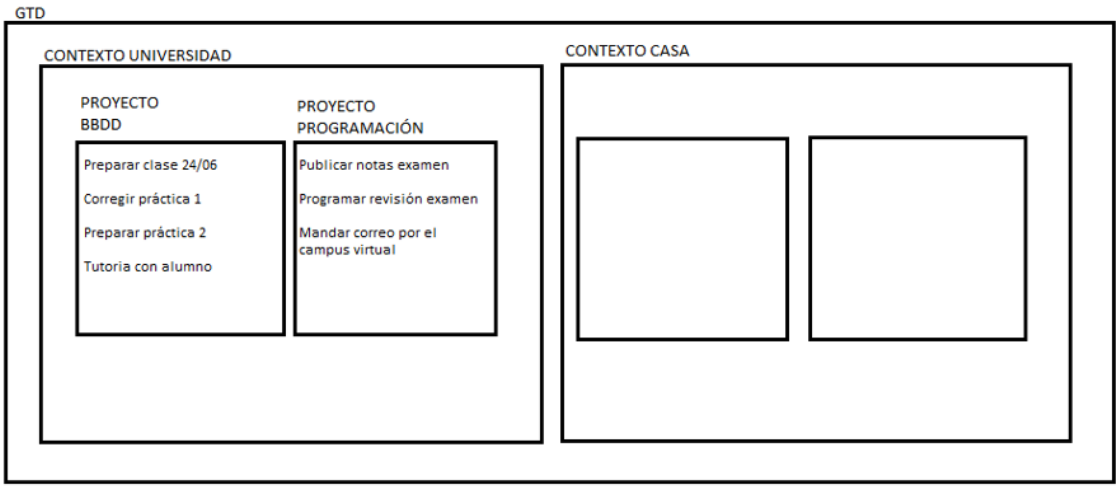

Figura 2.1: Esquema de ejemplo de la metodología GTD

2.3. Implementación GTD en Tracks

Tracks es una aplicación web (ver figura 2.2) que te permite aplicar la filosofía GTD recopilando tus tareas en la red.

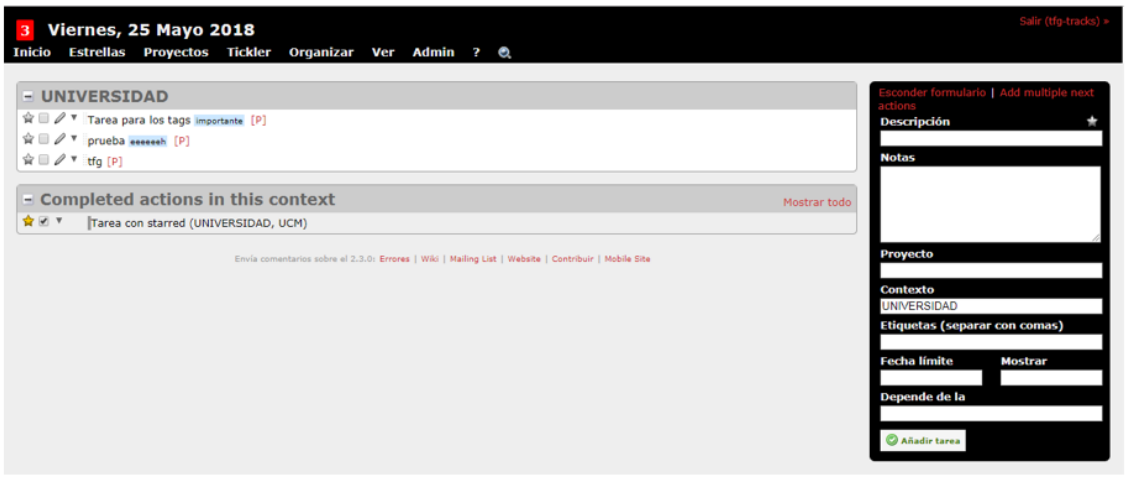

Figura 2.2: Captura de la aplicación web de Tracks

El menú de la barra superior está diseñado de tal manera que recoge los principales conceptos de GTD:

Las pestañas de **Inicio** y **Tickler** se corresponden con las tareas. Tracks nos muestra las tareas que tenemos pendientes organizadas por contextos. ¿En qué se diferencian entonces Inicio y Tickler? En la prioridad que nosotros mismos le hayamos dado a la tarea en función de la fecha, es decir, la pantalla Inicio nos mostrará las tareas que:

- No tengan ni una fecha límite para realizarse ni una fecha en la que queremos que nos recuerden que tenemos que realizar la tarea
- Tienen una fecha limite pero no una fecha en la que tracks nos recuerde que la tenemos que hacer.

Por el contrario, tickler nos mostrará todas aquellas tareas que:

- Tengan una fecha límite para hacerse o una fecha en la que queremos que nos recuerde hacer la tarea
- No tengan fecha límite para realizarse pero si fecha en la que recordarnos que hay que

hacerla.

La pantalla **estrellas** nos mostrará organizadas por contextos todas aquellas tareas que ya hayamos comenzado a realizar. **Proyectos** nos mostrará todos los proyectos que tenemos creados y el número de tareas que tiene asociado cada uno de ellos. La pestaña Proyectos clasifica los proyectos según su estado: activo, escondido o completo. Al igual que los proyectos, los contextos aparecen a través de la pestaña Organizar. Se muestran también según su estado y con el número de tareas que tienen asociados. Por último, la denominada **checklist** según Allen aparece a través de la pestaña **Ver** y en **Hecho**. Los proyectos y las tareas cuyo estado sea completado se mostrarán ahí.

Capítulo 3

Tecnologías utilizadas

A lo largo de este capítulo se describirán las tecnologías utilizadas para desarrollar la infraestructura GTD multiplataforma: Python, Pip, Kivy, kivy-garden, kivy-desginer, Buildozer y Request. Para todas ellas se proporcionará una pequeña introducción, una descripción sobre su propósito, su instalación y los contratiempos con los que nos hemos encontrado al trabajar con ellas.

Las cinco primeras tecnologías aparecerán dos veces debido a que han sido utilizadas tanto en Windows como en Ubuntu.

3.1. Entorno Windows

El sistema operativo que íbamos a utilizar inicialmente para desarrollar la aplicación era Windows debido a que tanto, por temas de trabajo como de ocio, es la plataforma con la que estamos más familiarizados.

Los primeros pasos que seguimos fue la instalación de Python y PIP en nuestros equipos:

3.1.1. Python

Python es un lenguaje de programación, cuyo principal interés es la programación de un código legible. Para su instalación seguimos los siguientes pasos:

- Desde la página oficial de Python (`https://www.python.org/`) nos descargamos la versión 2.7.14 y obtuvimos un archivo .msi

- Al abrir el archivo, comenzó su instalación y lo único que tuvimos que hacer fue seguir los pasos de su guía.
- Finalizada la instalación, para asegurarnos de que Python estaba instalado en nuestro equipo, desde un terminal introducimos el siguiente comando "`python --version`", como se puede ver en la figura 3.1.

Figura 3.1: Versión de Python instalada

3.1.2. PIP

PIP es una aplicación de línea de comandos que se utiliza para instalar y administrar paquetes de Python. Para su instalación, seguimos los pasos descritos a continuación:

- Descargamos el siguiente archivo "get-pip.py" (`https://pip.pypa.io/en/stable/installing/`).
- Abrimos una consola de comandos en modo administrador en la ruta en la que descargamos el archivo y ejecutamos el siguiente comando:

```
python get-pip.py
```

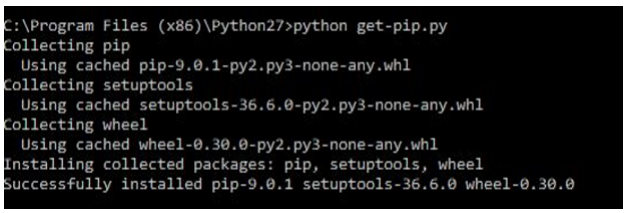

Figura 3.2: Proceso de instalación de PIP

3.1.3. Kivy

Una vez instalados en nuestro equipo Python y Pip, procedemos a la descarga e instalación de Kivy para comenzar a trabajar. Kivy es un framework open-source para Python que permite desarrollar aplicaciónes multiplataforma. Los pasos que seguimos fueron los siguientes:

- Instalamos las dependencias necesarias para Kivy con el siguiente comando:

```
python -m pip install docutils pygments pypiwin32
        kivy.deps.sdl2 kivy.deps.glew
```

- A continuación, procedemos a instalar Kivy:

```
python -m pip install kivy
```

Tras la instalación de Kivy, el siguiente paso era proceder a instalar Kivy Garden, el cual nos iba a proporcionar un conjunto de complementos (librerías) que nos ayudarían a la hora de desarrollar nuestra interfaz, y el mencionado anteriormente Kivy Designer.

3.1.4. Kivy-garden

Kivy Garden es un proyecto para centralizar los complementos para Kivy mantenidos por una comunidad de desarrolladores. Hoy en día se distribuye como un módulo aparte de Python y se instala de la siguiente manera:

```
pip install kivy-garden
```

3.1.5. Kivy-designer

Kivy Designer es la herramienta gráfica basada en widgets de Kivy para diseñar interfaces de usuario (GUI) de Kivy. Puede componer y personalizar widgets y probarlos. Está escrito completamente en Python usando Kivy. Los pasos que seguimos para la instalación de kivy-designer fueron los siguientes:

- Descargar o clonar el código de kivy-Designer del siguiente repositorio (`https://github.com/kivy/kivy-designer/archive/master.zip`)

- Extraer el archivo zip
- Entrar en la carpeta `kivy-designer`
- Ejecutar el siguiente comando:

```
pip install -Ur requirements.txt
```

- Finalizada la instalación ejecutamos el archivo `kivy.bat` para terminar de completar la instalación.
- Terminada la ejecución del archivo .bat, tan solo tendremos que ejecutar el siguiente comando para iniciar kivy-designer.

```
python -m designer
```

Una vez que teníamos instalados todos los componentes que necesitábamos, procedimos a tener una primera toma de contacto con Kivy, pero aquí comenzó uno de los muchos problemas que íbamos a tener durante el desarrollo del Trabajo Fin de Grado.

Cuando intentábamos iniciar kivy-designer con el comando proporcionado en la propia página de Kivy, ésta no se iniciaba y en la consola de comandos teníamos una larga traza de errores. Tras encontrarnos un poco perdidos, ya que nuestra herramienta principal no se iniciaba, decidimos hacer un cambio repentino y comenzar a utilizar Ubuntu sobre una máquina virtual. Para ello procedimos a la instalación de Virtual Box y de una imagen de disco de Ubuntu la cual descargamos del siguiente enlace: `http://releases.ubuntu.com/16.04/`.

Con Ubuntu instalado la máquina virtual, procedimos a volver a instalar todas las herramientas anteriormente citadas: Python, Pip, Kivy, kivy-garden y kivy-designer.

3.2. Entorno Ubuntu

En Ubuntu, la instalación de todas las tecnologías necesarias ya citadas anteriormente fue mucho más sencilla, como describiremos a continuación.

3.2.1. Python

Para la instalación de Python en Ubuntu tuvimos que ejecutar los siguientes comandos desde una consola de comandos:

```
sudo add-apt-repository ppa:fkrull/deadsnakes
sudo apt-get update
sudo apt-get install python2.7
```

3.2.2. PIP

Para la instalación del Pip tan solo fue necesario utilizar un comando:

```
sudo apt-get install python-pip
```

3.2.3. Kivy

Los pasos que seguimos para la instalación de Kivy en Ubuntu fueron los siguientes:

- Añadir el PPA de Kivy para futuras actualizaciones

  ```
  sudo add-apt-repository ppa:kivy-team/kivy
  ```

- Actualizar todos los paquetes que tengamos instalados:

  ```
  sudo apt-get update
  ```

- Instalar Kivy para la versión de Python 2.7:

  ```
  sudo apt-get install python-kivy
  ```

3.2.4. Kivy-garden

```
pip install kivy-garden
```

3.2.5. Kivy-designer

```
pip install -Ur requirements.txt
garden install filebrowser
python -m designer
```

Con todo instalado procedimos a iniciar kivy-designer (ver figura 3.3).

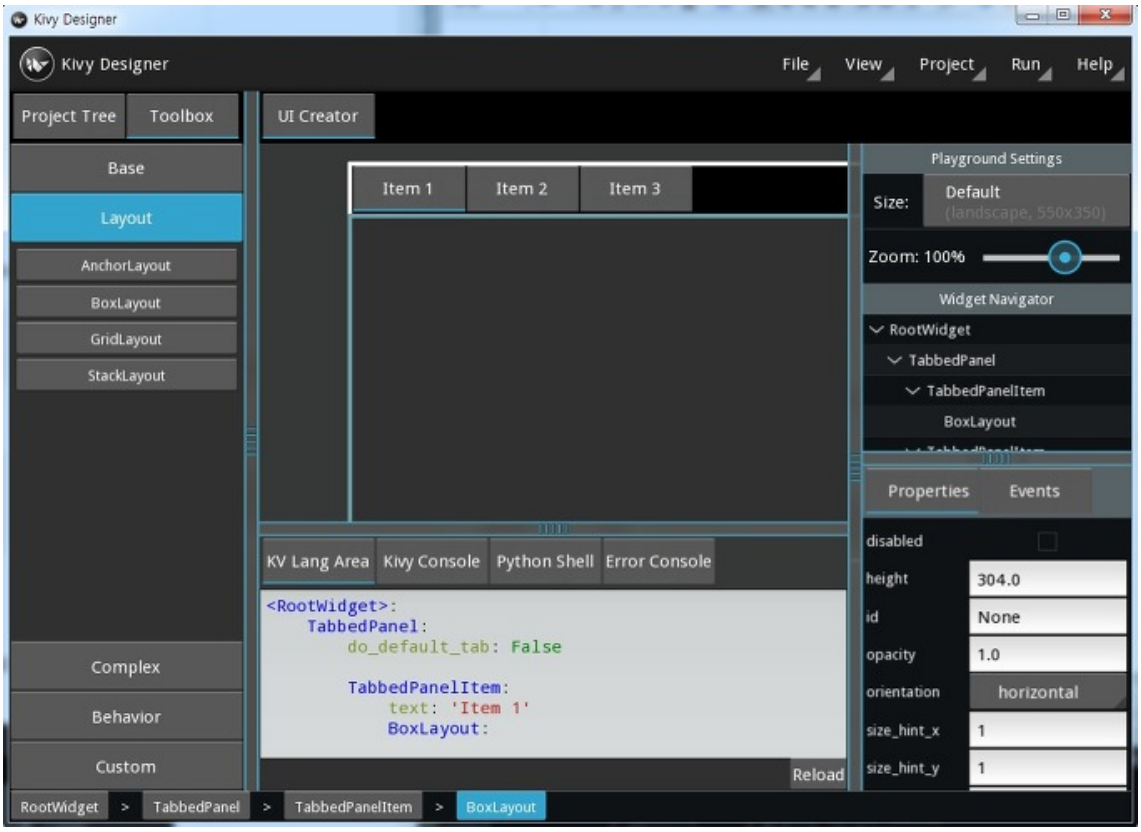

Figura 3.3: Captura del programa kivy-designer

Una vez instaladas las dependencias, podemos ejecutar el cliente en Ubuntu usando el siguiente comando:

```
python main.py
```

3.3. Buildozer

Buildozer es una herramienta que permite generar un APK (paquete de aplicación de Android) que posteriormente se instala en un dispositivo Android. Buildozer automatiza todo el proceso de compilación, descarga los requisitos previos como python-for-android, Android SDK, NDK, etc.

Buildozer empaqueta de manera sencilla todos los archivos necesarios para hacer funcionar la aplicación. A través del archivo de configuración buildozer.spec situado, en el directorio de la aplicación, se describen los requisitos y la configuración de la aplicación, como título, icono, módulos incluidos, etc.

3.3.1. Instalación

La instalación de buildozer se realiza de manera sencilla como descubriremos a continuación, pero su uso para la generación del archivo .apk, ha sido uno de los aspectos que más problemas nos ha dado durante la realización del trabajo. Los comandos para la instalación son los siguientes:

```
pip install --upgrade buildozer
sudo pip install --upgrade cython==0.21
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install build-essential \
        ccache git libncurses5:i386 libstdc++6:i386 \
        libgtk2.0-0:i386 libpangox-1.0-0:i386 \
        libpangoxft-1.0-0:i386 libidn11:i386 \
        python2.7 python2.7-dev openjdk-8-jdk \
        unzip zlib1g-dev zlib1g:i386
```

Una vez que teníamos Buildozer instalado en nuestro equipo, procedimos a hacer una prueba sencilla para generar el APK de una aplicación. La aplicación constaba de un simple botón que no hacía nada. El objetivo, no obstante, era asegurar que Buildozer funcionaba correctamente.

El primer paso que debíamos seguir era generar el archivo `buildozer.spec`, para ello hicimos uso del siguiente comando:

```
buildozer init
```

Generado el archivo `buildozer.spec`, ahora debemos rellenar algunos campos dentro del archivo, como el nombre de la aplicación, el nombre de los paquetes o la versión de la aplicación.

Con todos estos datos dentro de nuestro archivo, el siguiente paso es introducir el siguiente comando `buildozer -v android debug` para generar el APK. Lamentablemente aquí comenzaron los problemas, como se puede observar en la figura 3.4.

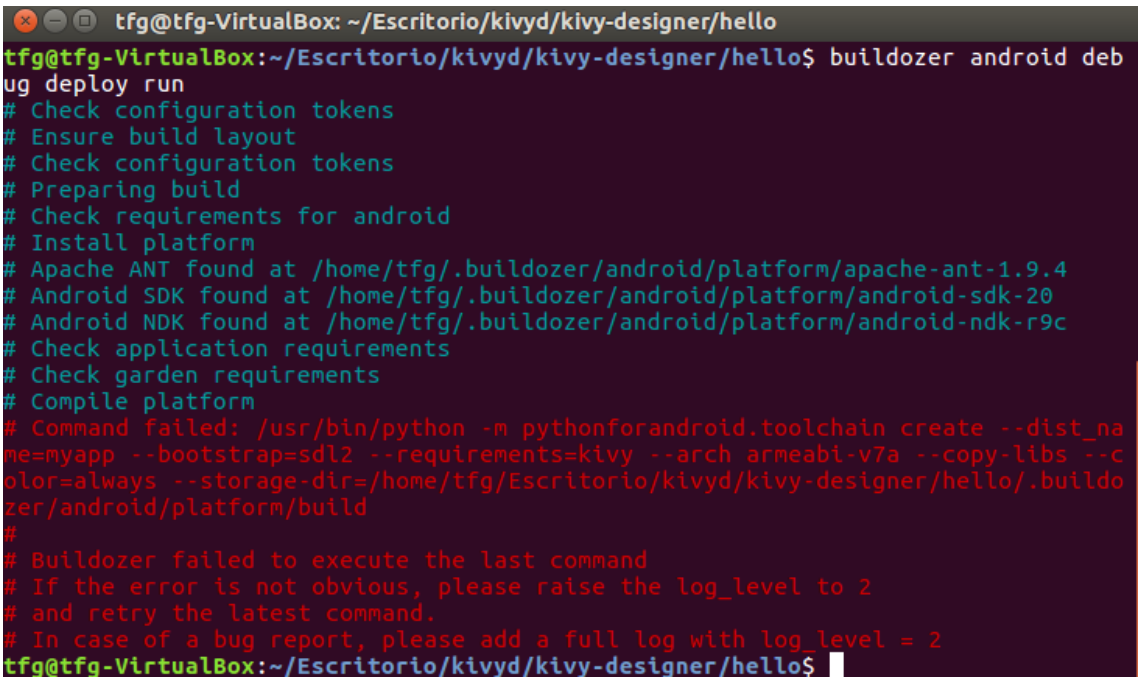

Figura 3.4: Error de compilación de Buildozer

La solución de este error nos llevó más tiempo del que creíamos, y conseguimos solucionarlo gracias al director de nuestro TFG. El problema principal era que las versiones que estábamos utilizando de muchos de los componentes no eran compatibles con la arquitectura para la que queríamos construir la aplicación, por lo que el APK nunca se generaba.

Tras cambiar la arquitectura para la que queríamos construir a armeabi-v7a, la versión de Android NDK a la 13b, y actualizar Cython a la versión 0.25, conseguimos generar el APK de la aplicación. Actualizar Cython de versión fue fácil, ya que tan solo tuvimos que utilizar el siguiente comando: `pip install Cython --install-option="0.25"`. Para la actualizacion de Android NDK tuvimos que investigar algo más hasta que encontramos en

qué ruta iba (`~/.buildozer/android/platform`). Una vez encontrada, tan solo tuvimos que descargar la versión 13b y descomprimirla con los siguientes comandos:

```
wget "https://dl.google.com/android/repository/android-ndk-r13b-linux.zip"
unzip android-ndk-r13b-linux-x86_64.zip -d /.buildozer/android/platform
```

A partir de este momento, para generar un nuevo APK tan solo teníamos que cambiar los datos que había en el archivo `buildozer.spec`. Estos datos hacían referencia al nombre y dominio de la aplicación (`package.name` y `package.domain`) y a las bibliotecas externas a Python que utilizase (requirements) para ejecutar el comando citado anteriormente: `buildozer -v android debug`

3.4. REST API

Para integrar nuestro cliente desarrollado en Python y Kivy (ver sección 4.2) y el servidor de Tracks (ver sección 4.1), precisamos de un mecanismo para leer y modificar nuestros datos. Para ello existen las REST API.

Un REST API es un servicio que mediante peticiones nos permite acceder a los datos de una aplicación o editarlos. La arquitectura REST (del inglés: Representational State Transfer) trabaja sobre el protocolo HTTP. Por consiguiente, los procedimientos o métodos de comunicación son los mismos que HTTP, siendo los principales: GET, POST, PUT, DELETE. Otro componente de un REST API es el "HTTP Status Code", que le informa al cliente o consumidor del API de qué debe hacer con la respuesta recibida.

Vamos a enumerar una serie de ventajas que ofrece REST para el desarrollo de un proyecto:

- **Separación entre cliente y servidor**: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos puedan evolucionar de forma independiente.

- **Visibilidad, fiabilidad y escalabilidad**: La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto

sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front-end y el back-end y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

- **La API REST siempre es independiente del tipo de plataformas o lenguajes**: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Hablaremos de la REST API de Tracks más adelante (ver sección 4.1.4).

3.4.1. Requests

Requests es un módulo de Python que nos permite realizar todo tipo de peticiones HTTP. Es una librería que aporta mucha funcionalidad, variando desde pasar parámetros en URLs a enviar encabezados personalizados y verificación SSL.

A continuación se muestra un ejemplo de petición GET (obtención de datos):

```
headers = {'Content-Type': 'text/xml'}
response = requests.get("https://kivy.dacya.ucm.es/projects.xml",
        auth = (connection.user, connection.passwd), headers = headers)
```

Toda la información sobre nuestra petición está ahora almacenada en un objeto `response` de tipo `Response`. Por ejemplo, se puede obtener la codificación de la página web usando la propiedad `response.encoding`. También se puede obtener el código de estado de la petición usando la propiedad `response.status_code`.

Ahora planteamos un ejemplo de edición de datos en el que podemos cambiar cualquier campo del objeto escribiéndolo en código XML. Incluimos los campos del objeto que queremos cambiar en la variable *data* y especificamos la URL sobre la que queremos ejecutar la petición:

```
    headers = {'Content-Type': 'text/xml'}
    data = "<project><name>"+proyecto.nombreProyecto+"</name>" +
            + "<description>"+proyecto.descripcion+"</description>"
            + "<default-tags>"+proyecto.tags+"</default-tags>"
            + "<default-context-id>"+proyecto.context+"</default-context-id>"
            + "</project>"
    response = requests.post("https://kivy.dacya.ucm.es/projects.xml",
            auth = (user, passwd), headers = headers, data = data)
```

Capítulo 4

Componentes de la infraestructura

En este capítulo se explican los diversos componentes que permiten el funcionamiento de nuestra infraestructura GTD multiplataforma. A continuación se muestra un esquema con una visión global de la infraestructura:

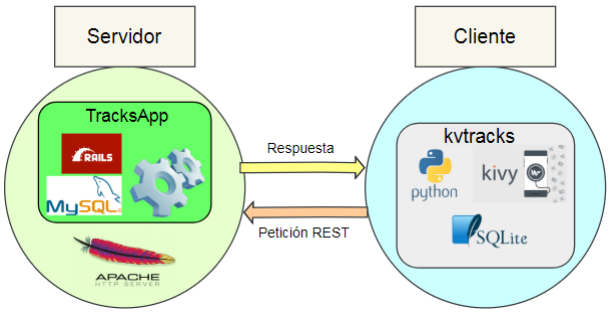

Figura 4.1: Esquema de la infraestructura GTD multiplataforma

4.1. Servidor

Nuestro servidor está compuesto por la aplicación web Tracks instalada en una máquina virtual con Linux como sistema operativo. Tracks es una aplicación Ruby on Rails, por lo que necesita Ruby para poder ejecutarse. Además, utilizamos Apache para poder acceder a la aplicación a través de HTTPS. Todos estos componentes los explicamos a continuación.

4.1.1. Máquina virtual

Para la instalación y ejecución de nuestro servidor, el director del TFG puso a nuestra disposición una máquina virtual alojada en servidores de la Universidad Complutense. El nombre del host es `kivy.dacya.ucm.es`, con usuario y contraseña personalizados para el acceso. Para acceder a la máquina se utiliza un protocolo denominado SSH (Secure SHell). Se trata de un protocolo de acceso remoto que nos ofrece un terminal seguro y encriptado a través del puerto 22.

Para acceder a nuestra máquina utilizamos la herramienta Putty (cliente SSH, ver figura 4.2), con la que podemos conectarnos a servidores remotos iniciando una sesión en ellos, y que permite ejecutar comandos.

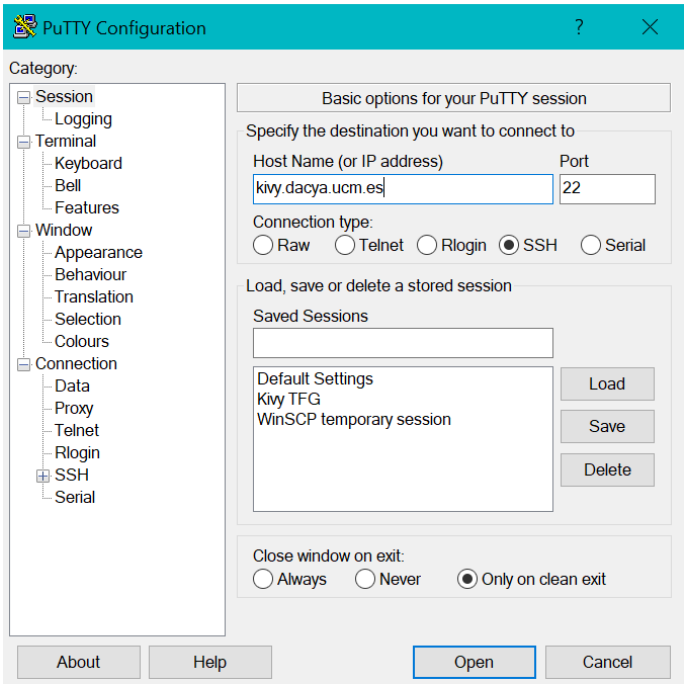

Figura 4.2: Captura del programa Putty que permite el acceso a la máquina por SSH

Como se ha comentado, esta máquina era propia de la UCM, lo que acarreaba diversos problemas. En ciertos momentos la máquina dejaba de estar disponibles y no podíamos levantarla porque no estaba en nuestro dominio, sucediendo varias veces a lo largo del curso. Al carecer de avisos sobre el estado de esta máquina, dependíamos del director del TFG para saber en qué momento tendríamos acceso, por lo que los retrasos se aumentaban

inutilizándonos durante algunos periodos de tiempo.

Podemos observar el proceso de petición de los datos para acceder a la máquina por SSH en la figura 4.3.

Figura 4.3: Captura de acceso a la máquina a través de Putty

Una vez dentro de la máquina disponemos de un sistema de ficheros vacío en el que copiaremos el código de Tracks con los archivos necesarios para su instalación. En esta máquina con Ubuntu 16.04.2 instalado podemos ejecutar comandos de Linux usando el Shell BASH.

Para el acceso a nuestra máquina nos proporcionaron un usuario con permisos de sudo que nos permite instalar y manipular la máquina para la realización de nuestro proyecto. Sin estos permisos, parte de la instalación de los diversos componentes que necesitamos no hubiese sido posible. Además, para que el usuario tenga la mejor experiencia de uso de nuestra aplicación, Tracks debe estar disponible las 24 horas del día.

La ejecución del comando nohup en el usuario root nos permite que nuestra aplicación Tracks esté continuamente levantada, facilitando al usuario el almacenamiento y la edición de las tareas, proyectos y contextos en cualquier momento del día.

4.1.2. Tracks

Tracks tiene una serie de requisitos de software que se especifican en la guía de instalación (ver cita [3]). Los requisitos principales son 3:

- **Ruby on Rails**: Para la instalación de Tracks se requiere Ruby on Rails versión 1.9.3 o mayor. Ruby on Rails (ver cita [15]) es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el patrón Modelo-Vista-Controlador (MVC). Para instalar Ruby on Rails, tuvimos que actualizar los repositorios de nuestra máquina con el siguiente comando:

```
sudo apt-get update
```

Con los repositorios actualizados, instalamos Ruby on Rails con este comando:

```
sudo apt-get install ruby-full
```

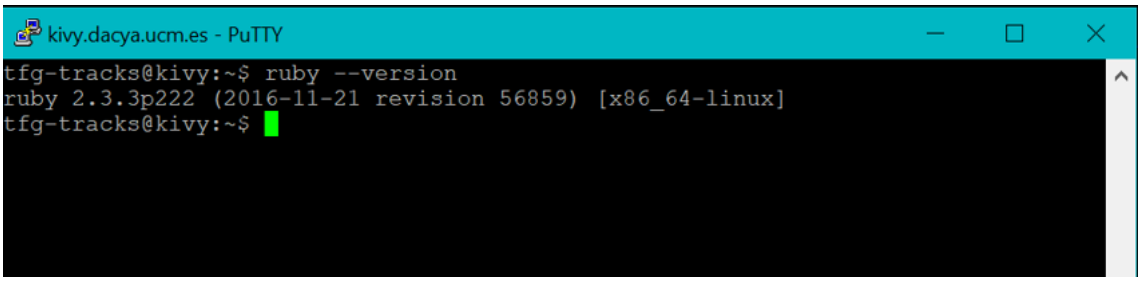

Figura 4.4: Comprobación de la instalación de Ruby

- **Bundler**: Tracks requiere la instalación de Bundler (ver cita [16]) en una versión reciente. Bundler es un manejador de dependencias para Ruby. Se instala con el siguiente comando:

```
gem install bundler
```

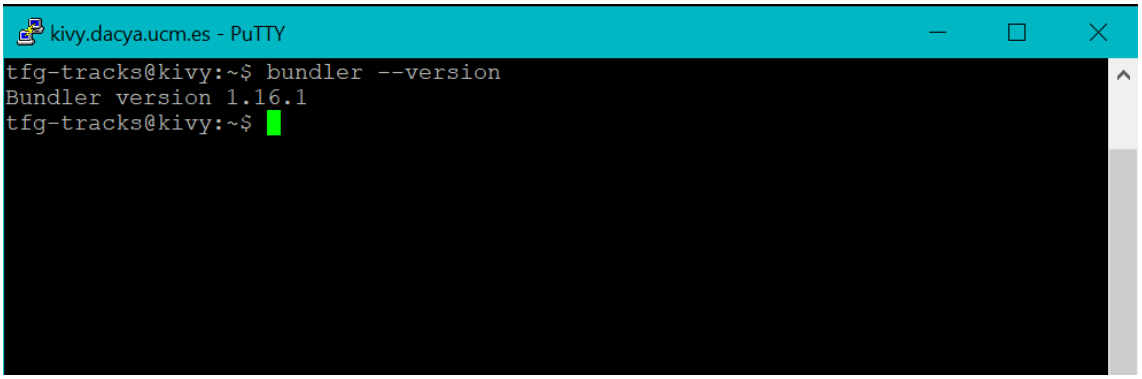

Figura 4.5: Comprobación de la instalación de Bundler

- **Base de datos**: Tracks requiere una base de datos que puede gestionarse con MySQL, SQLite y PostgreSQL. Como solo está testado en la primera, nos decidimos por MySQL (ver cita [17]). Para instalar MySQL utilizamos el siguiente comando:

```
sudo apt-get install mysql-server
```

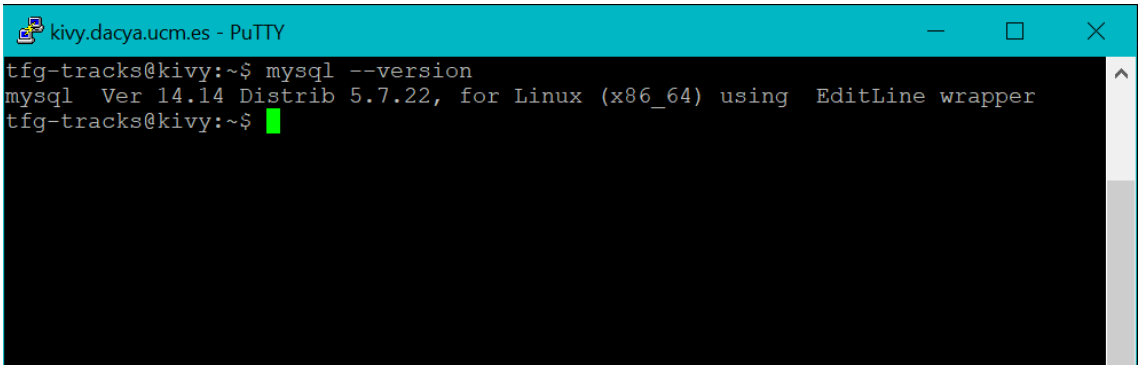

Figura 4.6: Comprobación de la instalación de MySQL

Tras instalar MySQL, el asistente nos pide una configuración de nuestra base de datos. La creamos con el nombre "tracks" y nuestro usuario "root". Los comandos necesarios son:

```
mysql -uroot -p
mysql> CREATE DATABASE tracks;
mysql> GRANT ALL PRIVILEGES ON tracks.* TO root@localhost \
IDENTIFIED BY '*****' WITH GRANT OPTION;
```

Ahora tenemos disponible nuestra base de datos para almacenar todos los proyectos, tareas y contextos de los diferentes usuarios de Tracks.

Para la instalación de Tracks en nuestra máquina nos descargamos los fuentes de su repositorio de Github (ver cita [4]), ejecutando el siguiente comando:

```
git clone https://github.com/TracksApp/tracks.git
```

Accedemos a nuestra carpeta y tendremos todos los archivos de Tracks, como se puede ver en la figura 4.7.

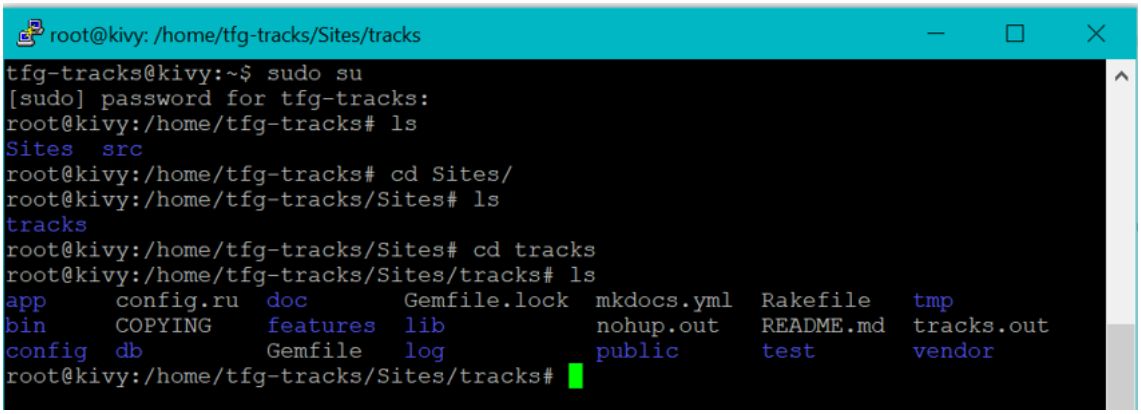

Figura 4.7: Listado de archivos de Tracks descargados de Github

Ahora que tenemos el código de Tracks descomprimido, procedemos a instalar sus dependencias. Ruby tiene un archivo llamado Gemfile con todas las dependencias que necesita instalar, y Tracks nos proporciona este archivo para que instale lo necesario para ejecutar la aplicación. Ejecutamos bundler para que instale todas las dependencias del Gemfile de esta forma:

```
bundle install --without development
```

Llegamos a la sección de configuración de variables para nuestra aplicación. Debemos modificar una serie de archivos que la guía nos indica:

- En la carpeta de configuración tenemos que editar los archivos `database.yml` y `site.yml`.
- Abrimos el archivo `database.yml` y editamos el apartado de producción (vamos a ejecutar Tracks como una aplicación orientada al entorno de producción). En este archivo configuramos los datos de nuestra base de datos que hemos creado en la sección anterior.
- Abrimos el archivo `site.yml` para cambiar la configuración de administrador y la zona horaria, adaptándola a la de nuestra máquina.
- Para configurar Tracks en un servidor web debemos cambiar la configuración de los estáticos de la aplicación (imágenes, estilos, etc.). En la ruta `/config/environments/`

`production.rb` cambiamos la variable `config.serve_static_assets` a `true`.

Para configurar correctamente nuestra base de datos con la estructura que Tracks requiere para guardar toda la información sobre proyectos, contextos y tareas, ejecutamos el siguiente comando:

```
bundle exec rake db:migrate RAILS_ENV=production
```

Tambien debemos compilar los estáticos (imágenes, archivos de estilos y javascript) antes de iniciar la aplicación. Para ello hay que ejecutar:

```
bundle exec rake assets:precompile RAILS_ENV=production
```

Ahora llega el momento de iniciar el servidor. Desde un shell de superusuario (root), hay que ejecutar el siguiente comando:

```
nohup bundle exec rails server -b kivy.dacya.ucm.es -p 3000
        -e production > tracks.out 2>&1 &
```

Los componentes del comando anterior tienen la siguiente semántica:

- `nohup` indica que no se pare el proceso cuando se cierre la consola. Seguirá ejecutándose continuamente y se levantará si la máquina cae. Podremos pararla nosotros manualmente.
- `bundle exec rails server` indica el inicio de la aplicación de Ruby on Rails.
- `-b` nos permite especificar la URL en la que levantamos nuestra aplicación, en nuestro caso `kivy.dacya.ucm.es`.
- `-p` nos permite configurar el puerto en el que se levanta la aplicación. Elegimos el recomendado por Tracks, el 3000.
- `-e` indica el entorno en el que levantamos Tracks. Utilizamos el entorno de producción (production).
- `tracks.out 2>&1` redirige la salida de trazas de información, ya sea error o simple ejecución, al archivo `tracks.out`.

- `&`: este comando indica que la aplicación se ejecuta en segundo plano, por lo que podremos seguir utilizando la consola mientras Tracks está disponible.

Con todo esto la aplicación web de Tracks estará disponible en la URL `http://kivy.dacya.ucm.es:3000`, y podremos proceder con el registro y acceder a su funcionalidad. También estará disponible una versión móvil en `http://kivy.dacya.ucm.es:3000/mobile`.

Como comentamos en la introducción (ver sección 1.1), necesitamos que los datos de los usuarios estén seguros en la nube. Tracks, por defecto, tiene una opción para configurar la aplicación y que sólo permita el acceso por HTTPS. Esta opción se activa en el archivo ubicado en `/home/tfg-tracks/Sites/tracks/config/environments` llamado `production.rb` (porque lo ejecutamos en el entorno de producción). Se debe cambiar la variable `force_ssl` a `true`, redireccionando toda petición HTTP a HTTPS. Al realizar este cambio y ejecutar de nuevo Tracks, la aplicación dejó de estar accesible. Todas las páginas de Tracks daban error al intentar acceder de forma segura. Dado que en la documentación de Tracks (ver cita [3]) no se proporciona una solución a este problema, tuvimos que buscar una solución alternativa para ofrecer una conexión segura y encriptada.

4.1.3. Apache

Tras una búsqueda exhaustiva en Internet, vimos que existía la opción de usar un servidor web para servir nuestra aplicación en Ruby on Rails con certificado HTTPS. Este servidor web necesita Apache2 y Passenger (ver cita [8]), además de un certificado SSL para poder encriptar las peticiones.

Para obtener un certificado SSL tuvimos que ejecutar una serie de comandos en nuestra máquina. Estos comandos permiten obtener unos archivos `.key` y `.crt` con nuestras claves para generar el certificado, proporcionando algunos datos como el país, la ciudad, nombre de la organización, etc. Obtenermos estos ficheros con el siguiente comando:

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048
        -keyout /etc/ssl/private/apache-selfsigned.key
                -out /etc/ssl/certs/apache-selfsigned.crt
```

Con estas claves podemos generar nuestro certificado con el siguiente comando:

```
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

Una vez tengamos el certificado, debemos configurar nuestro servidor Apache2 para que utilice SSL con nuestro certificado y ejecutar nuestra aplicación Tracks. Esto lo realizamos editando el archivo ubicado en `/etc/apache2/sites-available`, con el nombre de `default-ssl.conf`:

```
<IfModule mod_ssl.c>
  <VirtualHost _default_:443>
    ServerAdmin adrimont@ucm.es
    DocumentRoot /home/tfg-tracks/Sites/tracks/public

    RailsEnv production
    ErrorLog \${APACHE_LOG_DIR}/error.log
    CustomLog \${APACHE_LOG_DIR}/access.log combined
    #   SSL Engine Switch:
    #   Enable/Disable SSL for this virtual host.
    SSLEngine on
    SSLCertificateFile      /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
    <FilesMatch "\.(cgi|shtml|phtml|php)\$">
                    SSLOptions +StdEnvVars
    </FilesMatch>
    <Directory "/home/tfg-tracks/Sites/tracks/public">
            Options FollowSymLinks
             Require all granted
    </Directory>
    <Directory /usr/lib/cgi-bin>
                    SSLOptions +StdEnvVars
    </Directory>
    BrowserMatch "MSIE␣[2-6]" \
                   nokeepalive ssl-unclean-shutdown \
                   downgrade-1.0 force-response-1.0

  </VirtualHost>
</IfModule>
```

Así, nuestro servidor Apache con HTTPS ubicado en la URL `https://kivy.dacya.ucm.es` nos redireccionará internamente a Tracks sin afectar al funcionamiento de nuestra aplicación, realizando una conexión totalmente segura y encriptada.

4.1.4. REST API

La API REST de Tracks permite a los desarrolladores integrar Tracks con clientes. Los clientes serán capaces de acceder y modificar datos del servidor.

La API es un servicio RESTful. Todos los datos están disponibles a través de la API como un recurso al que se puede acceder utilizando un identificador único. Se pueden utilizar varios métodos HTTP, específicamente GET, PUT, POST y UPDATE, y todas las respuestas de la API están en un formato XML.

La autenticación se maneja usando HTTP básico. El nombre de usuario y contraseña de Tracks se utilizan como credenciales de autenticación para la API. La aplicación de Tracks, al tener autenticación HTTP básica, nos indica que la contraseña de nuestro usuario se manda en texto plano. No ofrece ninguna solución, pero este problema lo solventamos al construir un servidor por encima de nuestra aplicación.

4.1.5. Testeo REST API Tracks

Tuvimos que realizar un testeo de la REST API para cerciorarnos de que ésta funcionaba correctamente. Para testear el envío y la recepción de datos y toda la funcionalidad que requiere la aplicación, debemos testear tres casos por cada tipo de objeto (proyecto, contexto y tarea). Para ello, ejecutamos los ejemplos que encontramos en la página de Tracks.

Obteniendo datos de Tracks

Para obtener datos de la aplicación web realizamos una petición con el comando `curl`:

```
curl -u user:pass -H "Content-Type:text/xml" \
  "http://kivy.dacya.ucm.es:3000/projects.xml"
```

Debemos especificar nuestro usuario y contraseña, con la cabecera indicando que el contenido está en XML, y dirigido a nuestra ruta obteniendo los proyectos (`/projects.xml`)

La respuesta que nos ofrece muestra los proyectos en lenguaje XML, con las distintas variables como el nombre, la descripción, el estado, etc.

Enviando datos a Tracks

Nos encontramos con tres acciones que podemos realizar sobre los datos localizados en Tracks:

- POST (crear objeto)

  Ejecutamos el siguiente comando para crear un nuevo objeto:

```
curl -u user:pass -H "Content-Type:text/xml" \
  -d "<project><name>EXAMPLE</name></project>" \
  "http://kivy.dacya.ucm.es:3000/projects.xml"
```

  Para el testeo, en el campo -d especificamos en lenguaje XML el nombre de nuestro nuevo proyecto. El resultado nos ofrece un ID en el campo *LOCATION* y nos indica que se ha creado correctamente con el código 201.

- POST-DELETE (eliminar objeto)

  Para testear la eliminación de un objeto, ejecutamos un nuevo POST, pero especificando que vamos a borrarlo con el comando -X DELETE. Para que se ejecute bien la eliminación debemos añadir a la URL del servidor a qué tipo de objetos pertenece y su ID:

```
curl -u user:pass -H "Content-Type:text/xml" \
  -X DELETE "http://kivy.dacya.ucm.es:3000/8.xml"
```

  La respuesta nos indica, con el código 200 y el mensaje "OK", que se ha borrado correctamente.

- POST-PUT (editar objeto)

  Para testear la edición de un objeto, debemos especificar el comando -X PUT e indicar con -d los datos que queremos editar. En este caso editamos el nombre del proyecto:

```
curl -u user:pass -H "Content-Type:text/xml" \
  -X PUT -d "<project><name>EXAMPLE</name></project>" \
  "http://kivy.dacya.ucm.es:3000/85.xml"
```

  La respuesta es correcta, indicándonos un código 200 y el mensaje "OK".

4.2. Cliente

Nuestro cliente es una aplicación multiplataforma desarrollada en Python y Kivy (ver citas [10] y [5]) llamada **kvtracks**. Como comentamos en la sección 1.2.2, hemos utilizado

Kivy para poder ejecutar el cliente en cualquier plataforma. Hemos comentado en el capítulo 3 las tecnologías del proyecto, además de la forma en la que generamos el APK, pero a continuación vamos a mostrar unas capturas donde vemos el cliente ejecutándose en Ubuntu (figura 4.8) y macOS (figura 4.9):

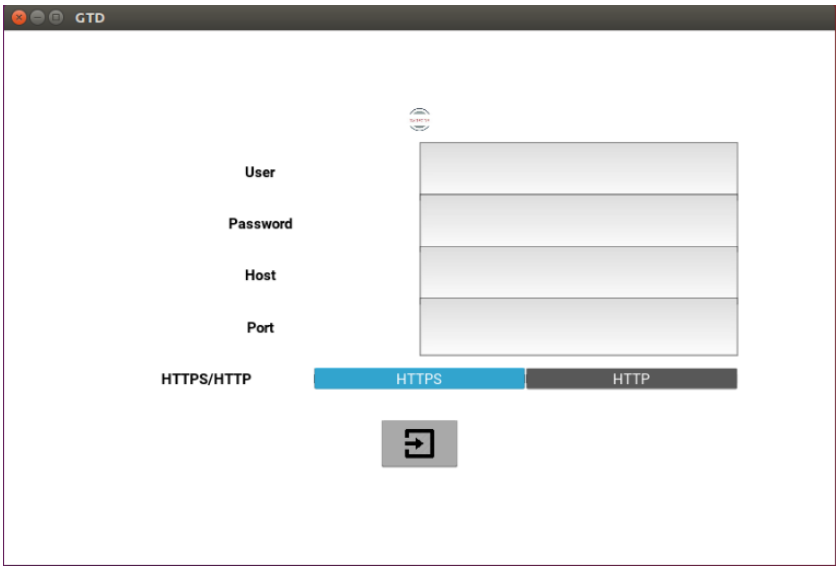

Figura 4.8: kvtracks ejecutándose en Ubuntu

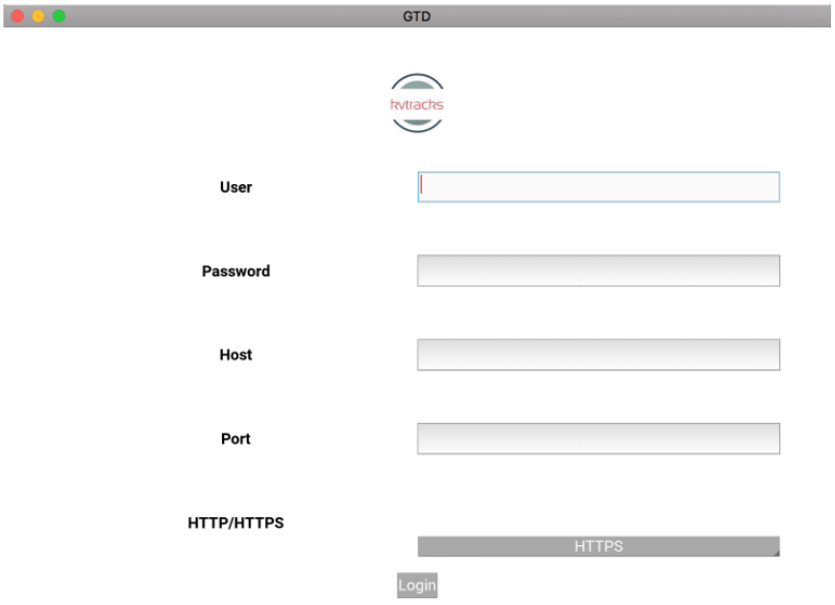

Figura 4.9: kvtracks ejecutándose en macOS

A continuación se explican los componentes del cliente kvtracks:

4.2.1. Backend y Frontend

El *backend* y el *frontend* son las capas que componen la aplicación. Están implementadas en Python y Kivy. El *backend* es la capa de acceso a datos, y sus componentes se explicarán a continuación de esta sección. El *frontend* es la capa de presentación e incluye la interfaz de usuario. Se hablará de la interfaz en el capítulo 5.

4.2.2. Base de datos

Como se comentó en la sección 1.2.3 nuestra aplicación dispone de un modo offline para que el funcionamiento no se vea afectado en los momentos en que no haya conexión a internet. Este modo sólo puede ser posible si se almacenan los datos en el propio dispositivo, y para ello se precisa de una base de datos. Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Existen programas denominados sistemas gestores de bases de datos que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Hemos optado por SQLite (ver cita [9]) como sistema gestor ya que es el único que acepta Kivy para la compilación de las aplicaciones.

SQLite es una herramienta de software libre que permite almacenar información de forma sencilla y rápida, en equipos con pocas capacidades de hardware. Se puede usar tanto en dispositivos móviles como en sistemas de escritorio, sin necesidad de realizar procesos complejos de importación y exportación de datos.

A continuación se muestra un ejemplo de uso de SQLite desde Python:

```
import sqlite3
con = sqlite3.connect('example.db', check_same_thread=False)
c = con.cursor()

# Create table
c.execute('''CREATE TABLE IF NOT EXISTS users
             (user text, passwd text)''')
```

Figura 4.10: Ejemplo de creación de base de datos SQLite con una tabla

Importamos la librería de SQLite, conectamos con nuestra base de datos *example.db* y con el cursor ejecutamos una creación tabla llamada *users*, que contiene los campos *user* y *passwd* ambos de tipo text. Sobre esta tabla podremos realizar diversas acciones como la inserción de nuevos elementos (INSERT), el borrado de elementos (DELETE), la edición de uno o varios elementos (UPDATE) o simplemente consultas sobre los datos de las tablas que hemos creado (SELECT).

Los datos de nuestros proyectos, contextos y tareas, los usuarios y sus contraseñas, y los objetos a crear, eliminar o guardar en la web se almacenan en estas bases de datos para que, una vez la conexión a internet vuelva a estar disponible, se realice el procedimiento. El usuario no necesitará saber si la conexión está disponible o no y siempre tendrá disponibles sus datos.

4.2.3. Orientación a objetos en kvtracks

La programación orientada a objetos es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial. Los objetos que componen nuestro proyecto son: contextos, proyectos y tareas.

- **Contextos**: Los contextos son objetos que contienen un identificador (ID), un nombre del contexto (es obligatorio) y un estado ("active" por defecto). Son los objetos con los que se asocian los proyectos y tareas poder clasificarlos.

  Un ejemplo de contexto podría ser FACULTAD DE INFORMÁTICA.

- **Proyectos**: Los proyectos contienen un ID, un nombre de proyecto (obligatorio), un contexto, una o varias etiquetas y un estado ("active" por defecto). Los proyectos se ubican en un contexto determinado, y las tareas se asocian a estos proyectos para que las podamos ubicar.

  Un ejemplo de proyecto podría ser TRABAJO DE FIN DE GRADO.

- **Tareas**: Las tareas contienen un ID, una descripción de la tarea, un contexto determinado, un proyecto, una fecha de vencimiento, una fecha para aparecer en el apartado TICKLER, una o varias etiquetas y un estado ("active" por defecto).

Un ejemplo de tarea podría ser HACER LA MEMORIA.

Este es un esquema que muestra la organización de la base de datos del cliente:

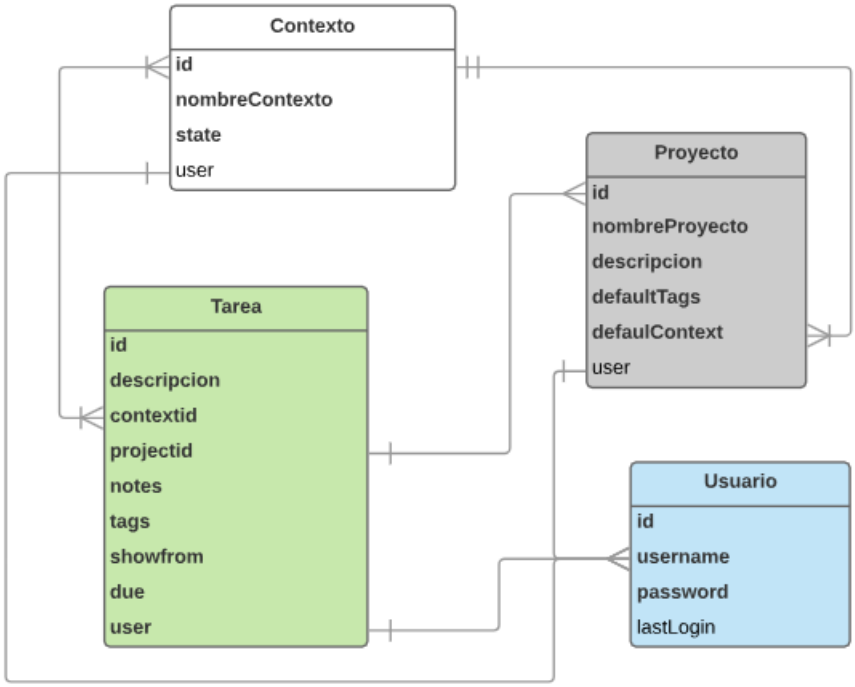

Figura 4.11: Esquema base de datos del cliente

4.2.4. Singleton

Singleton es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella. Como sólo disponemos de una conexión, hemos aplicado este patrón en nuestro cliente.

Los objetos de nuestra aplicación tendrán acceso a esta instancia y obtendrán los datos necesarios de conexión para la ejecución de sus tareas. Esta instancia dispone de 5 variables que configuran la conexión: *user*, *password*, *url*, *port* y *http/https*

4.2.5. Peticiones

Como vimos en la sección 3.4, para ejecutar las acciones de cada objeto en nuestra máquina utilizamos la librería de Python llamada Requests (ver cita [11]). Esta librería se

utiliza en cada objeto con dos funciones principales: cargar y guardar datos.

Las funciones de cargar y guardar son propias de cada objeto. Cuando se crea uno nuevo, se borra o se actualiza,se ejecuta en nuestro la función cargar o guardar para almacenar los datos en la nube. En el caso de que no haya conexión, los datos se almacenan en la base de datos del dispositivo hasta la próxima vez que esté disponible.

Capítulo 5

GUI (Interfaz de usuario) de kvtracks

La GUI o interfaz de usuario es la parte visible de la aplicación, que también se denomina frontend. A lo largo de este capítulo explicaremos las ideas que hemos tenido a la hora de diseñar la aplicación, y en qué otras aplicaciones, cuyo función/uso es similar a la de nuestra aplicación, nos hemos basado para diseñar **kvtracks**. Mostraremos desde los primeros bocetos que realizamos inicialmente, hasta los bocetos finales que pertenecen a la versión final de la aplicación.

En este capítulo también se incluye una guía de usuario en la que explicaremos la disposición de cada elemento de la GUI, y una serie de casos de uso básicos: creación, edición y eliminación de contextos, proyectos y tareas.

5.1. Motivación e inspiración

Una vez que todas las tecnologías documentadas en el capítulo 3 habían sido instaladas y se había comprobado la compatibilidad entre todas ellas con pequeñas prototipos, comenzamos a realizar distintos bocetos con la aplicación web balsamiq (ver cita [7]).

El principal objetivo a la hora de diseñar nuestra interfaz es que fuese una aplicación ante todo intuitiva. Para ello, construimos unos bocetos que muestran nuestro primer diseño de la GUI, que evolucionó a lo largo del tiempo. Esta evolución se debe al conocimiento creciente sobre la organización de contextos, proyectos y tareas como objetos, que se fue adquiriendo a lo largo del proyecto hasta una aproximación más cercana a la de la filosofía GTD explicada en el capítulo 2.

Los bocetos iniciales de la aplicación están recogidos en el apéndice C.

5.2. Guía de usuario

Al iniciar la aplicación, el usuario se encontrará con la pantalla de *login* en la que introduciendo el host y puerto al que se quiere conectar, su usuario y su contraseña podrá acceder al menú principal.

Tras iniciar sesión, el usuario se encontrará con un menú que divide claramente la aplicación en las distintas funcionalidades que tiene: TODAY, TICKLER, START, PROJECTS, ORGANIZE y SHOW DONE.

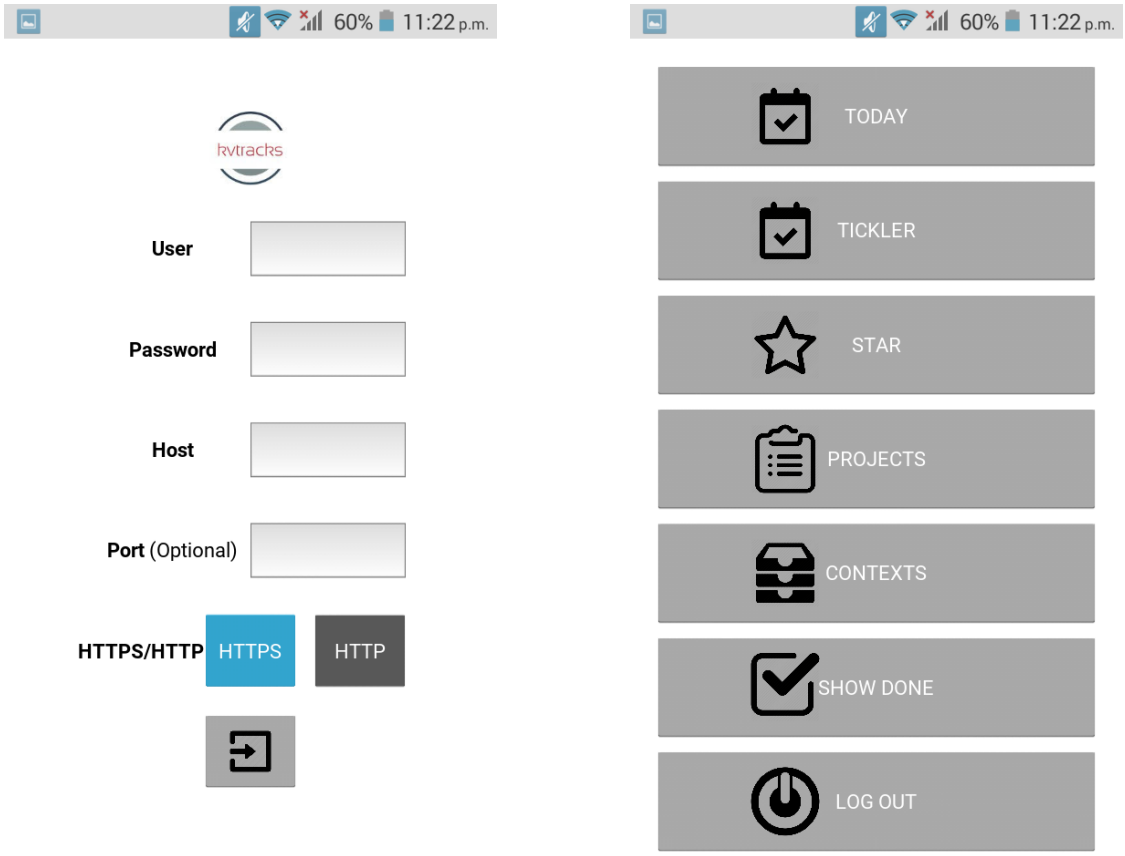

Figura 5.1: Login y pantalla principal

TODAY

La pantalla TODAY nos mostrará las tareas, organizadas por contextos, que tienen fecha límite para su realización y no fecha para ser mostradas, o que no tienen ni fecha límite ni fecha para ser mostradas.

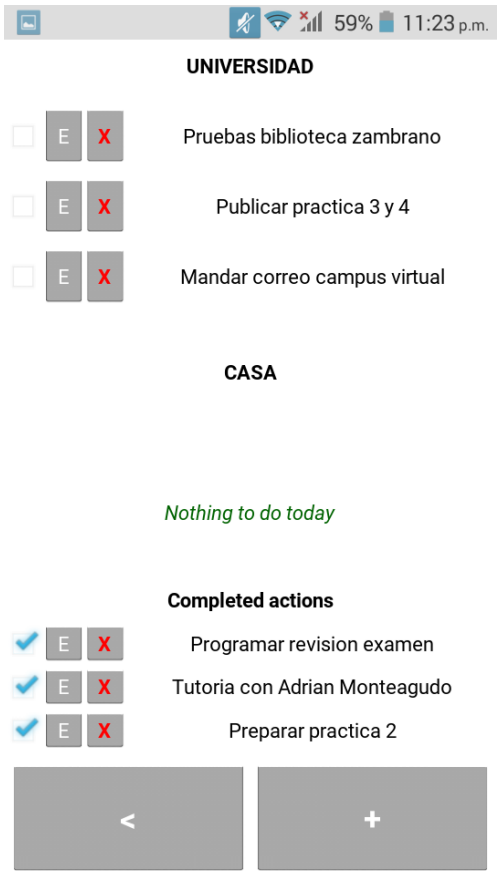

Figura 5.2: Pantalla TODAY

Como podemos ver, se muestra una lista de tareas organizadas por contextos. Cada tarea se puede eliminar (X) o editar (E). Además, abajo se muestran las tareas completadas, y dos botones con los que se puede volver atrás (<) o crear una nueva tarea (+).

TICKLER

La pantalla TICKLER nos mostrará las tareas, organizadas por contextos, que tienen fecha límite para su realización y fecha para ser mostradas, o que no tienen fecha límite pero sí fecha para ser mostradas.

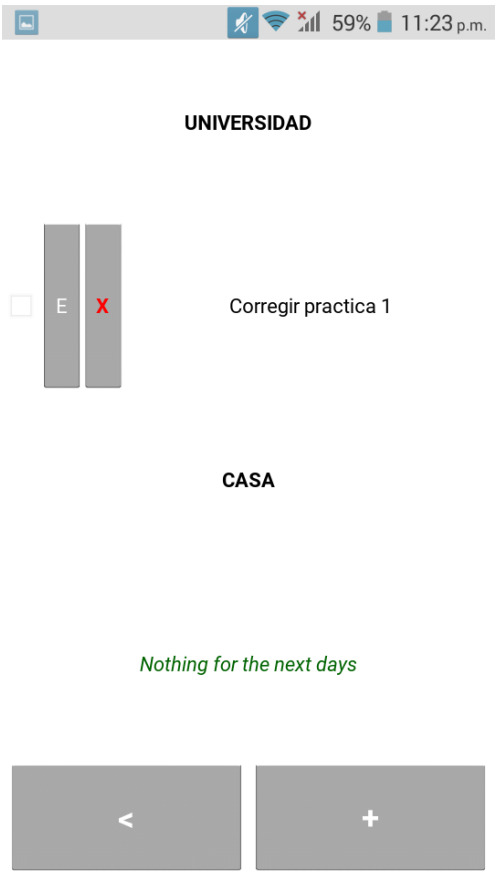

Figura 5.3: Pantalla TICKLER

Esta pantalla, al igual que TODAY, muestra las tareas organizadas por contextos. Se pueden editar, eliminar y crear nuevas tareas, además de volver a la pantalla principal.

STAR

La pantalla STAR nos mostrará las tareas, organizadas por contextos, que contienen el tag 'starred' (las marcamos con una estrella).

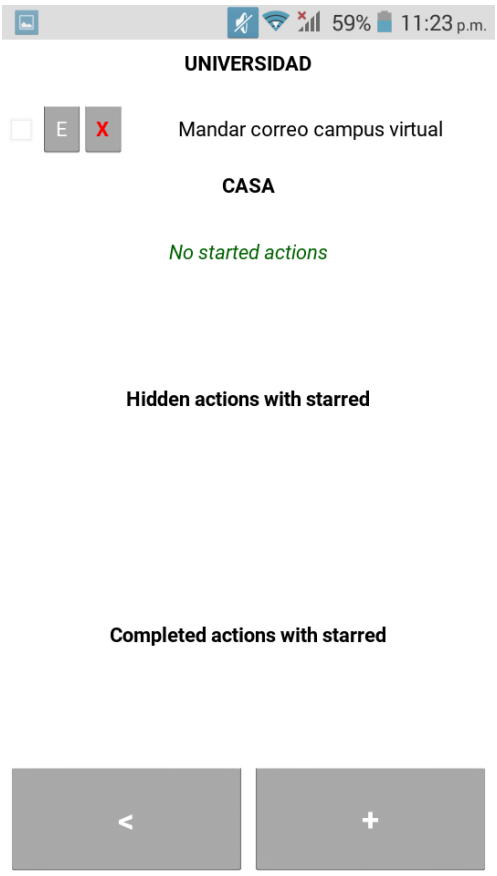

Figura 5.4: Pantalla STAR

Como se ve en la imagen, la pantalla es similar a las anteriores. Se muestran las tareas y se pueden editar y eliminar, o crear nuevas tareas. También podemos volver a la pantalla principal.

PROJECTS

La pantalla PROJECTS nos mostrará los proyectos organizados según su estado: `active`, `hidden` o `completed`.

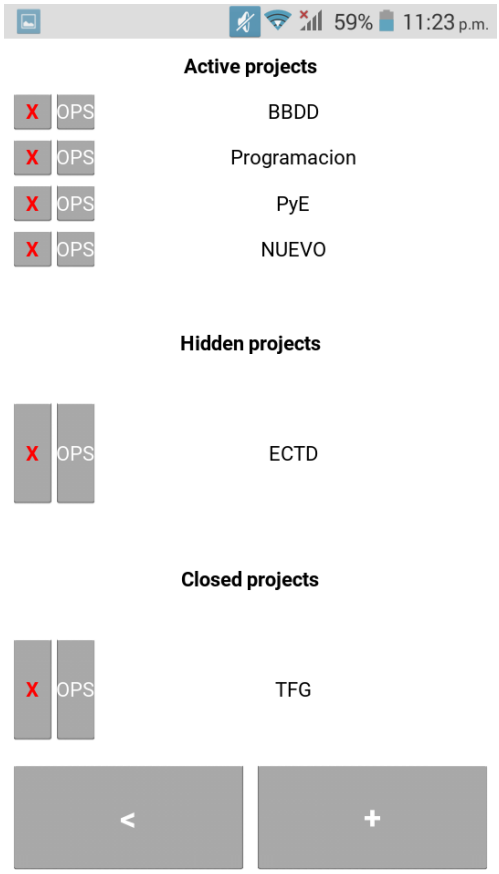

Figura 5.5: Pantalla PROJECTS

Se pueden ver los diferentes proyectos organizados por estado. Se pueden eliminar y editar, además de crear nuevos proyectos y volver a la página principal.

ORGANIZE

Al pulsar el botón ORGANIZE accederemos a la pantalla de los contextos organizados según su estado.

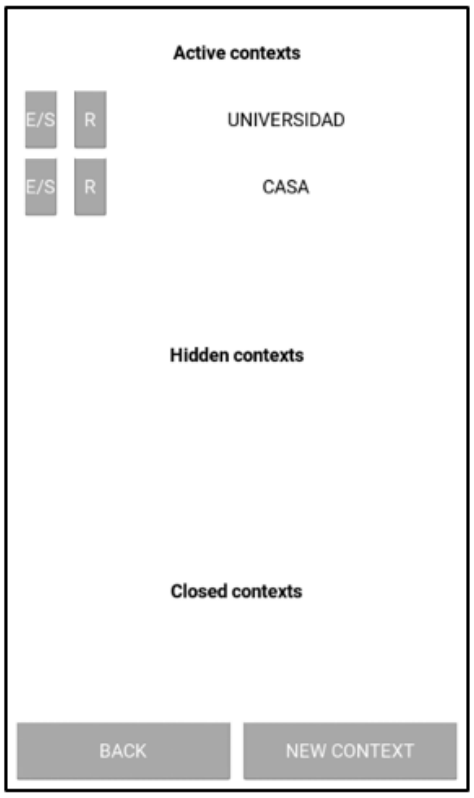

Figura 5.6: Pantalla ORGANIZE

Se pueden ver listados los contextos organizados por estado. Podremos editar o eliminar los contextos y crear nuevos gracias al botón *New Context*. También podemos volver a la pantalla principal.

SHOW DONE

La pantalla SHOW DONE nos mostrarán los proyectos y tareas que ya han sido realizados/finalidados, es decir, aquellos cuyo estado es `completed`.

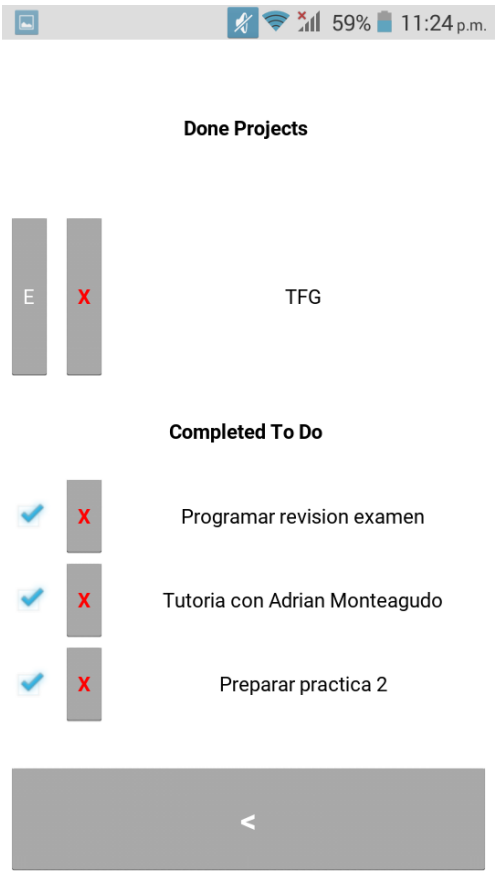

Figura 5.7: Pantalla SHOW DONE

Aquí se pueden ver los proyectos que se han cerrado y las tareas que están completadas. Podemos editar ambos objetos, y también hay un *check* a la izquierda de cada tarea que podemos desmarcar para cambiar el estado a activo.

5.2.1. Casos de uso

Siguiendo la metodología GTD, kvtracks nos permite crear, editar y eliminar contextos, proyectos y tareas.

Contextos

1. Crear un contexto

   Accediendo a la pantalla de Contextos y pulsando el botón de *New Context* (ver figura 5.6) accederemos a la siguiente pantalla que nos permitirá crear un nuevo contexto:

Figura 5.8: Crear contexto

   Completando el campo que nos aparece con el nombre del nuevo contexto y pulsando sobre el botón *Save*, el nuevo contexto quedará creado.

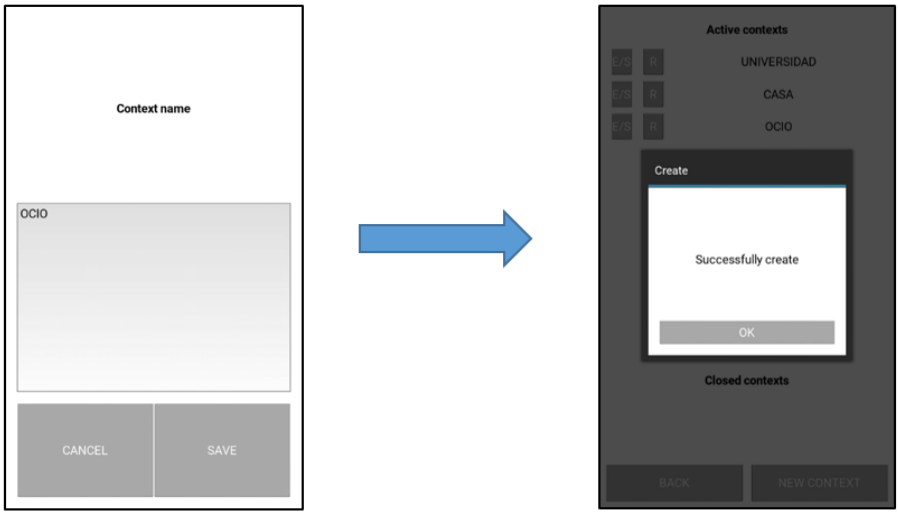

Figura 5.9: Contexto creado

2. Editar un contexto

   Accediendo a la pantalla de Contextos y pulsando el botón que contiene E/S (Edit/Show) accederemos a la siguiente pantalla que nos permitirá editar o ver las tareas asociadas al contexto elegido. Si pulsamos el botón *Save*, tras modificar el nombre del contexto se guardarán los resultados.

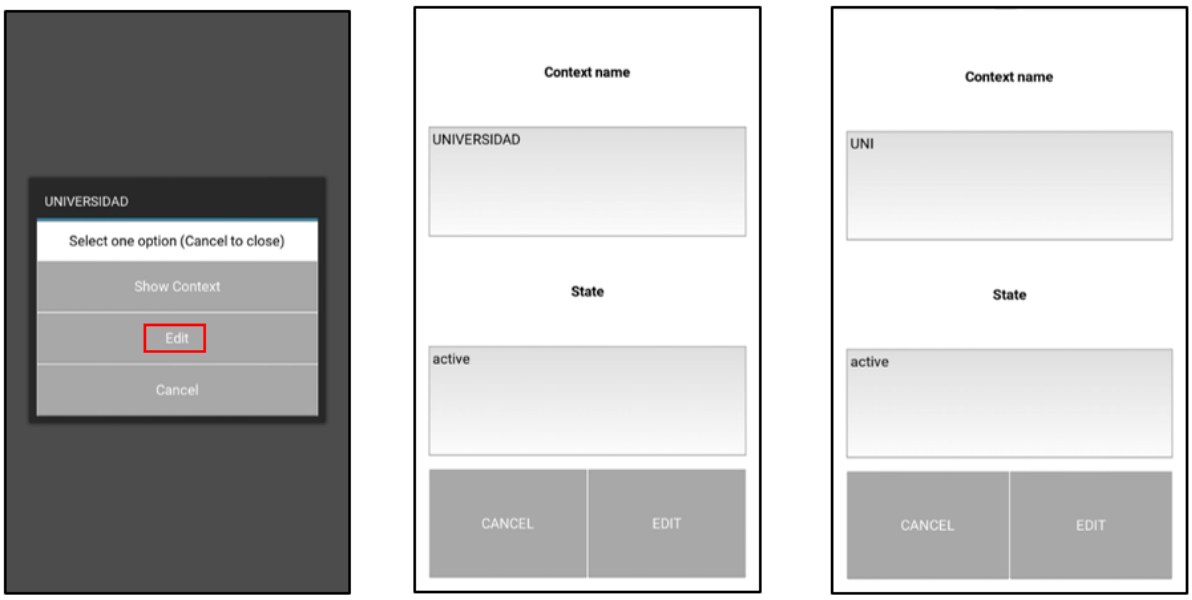

Figura 5.10: Editando un contexto

Si pulsamos el botón *Show Context* veremos las tareas asociadas a dicho contexto, como se puede ver en la figura 5.11:

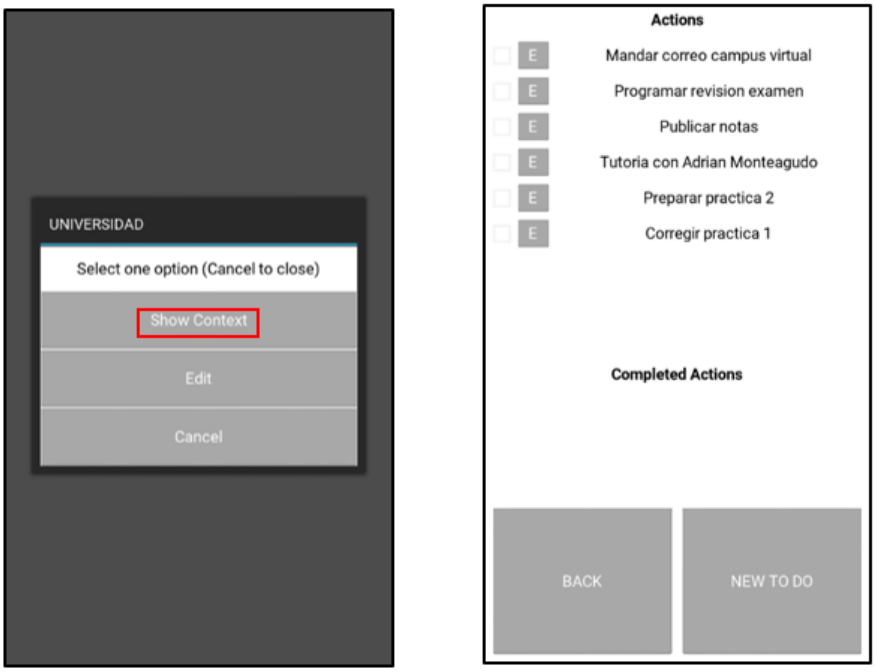

Figura 5.11: Mostrando las tareas de un contexto

3. Eliminar un contexto

Accediendo a la pantalla de Contextos y pulsando el botón que contiene una X se mostrará un cuadro de diálogo que nos permitirá eliminar el contexto.

Figura 5.12: Eliminando un contexto

Proyectos

1. Crear un proyecto

   Accediendo a la pantalla PROJECTS y pulsando el botón + accederemos a la siguiente pantalla que nos permitirá crear un nuevo proyecto. Pulsando el botón *Save* guardaremos el nuevo proyecto:

Figura 5.13: Creando un proyecto

2. Editar proyecto

   Accediendo a la pantalla PROJECTS y pulsando el botón E (Edit) accederemos a la siguiente pantalla que nos permitirá crear un editar y visionar los datos del proyecto que hayamos seleccionado. Pulsando el botón *Save* guardaremos los cambios en el proyecto, en caso de que hayamos realizado alguno.

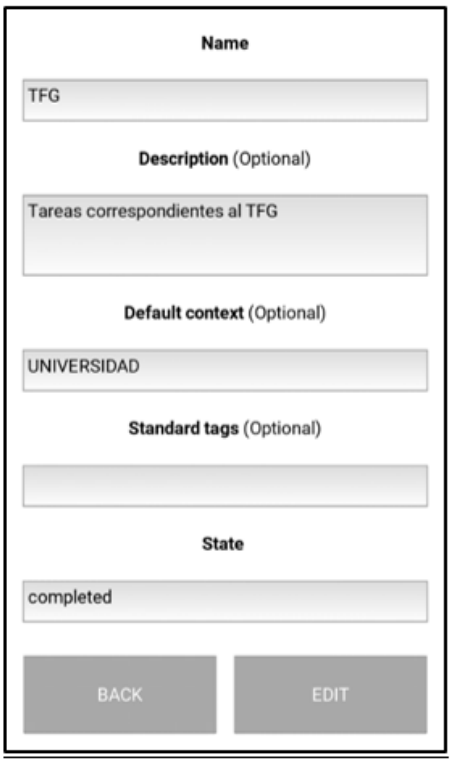

Figura 5.14: Editando un proyecto

3. Eliminar proyecto

   Accediendo a la pantalla PROJECTS y pulsando el botón X se mostrará un cuadro de diálogo que nos permitirá eliminar el proyecto que hayamos seleccionado.

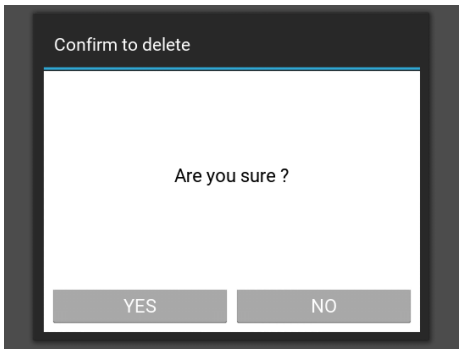

Figura 5.15: Eliminando un proyecto

Tareas

1. Crear una tarea

   Pulsando sobre el botón +, al que podremos acceder a través de las pantallas TODAY, TICKLER, STAR, o entrando en la información de un contexto, nos aparecerá la siguiente pantalla, en la que rellenando los datos que nos piden y pulsando sobre el botón *Save* guardaremos la nueva tarea:

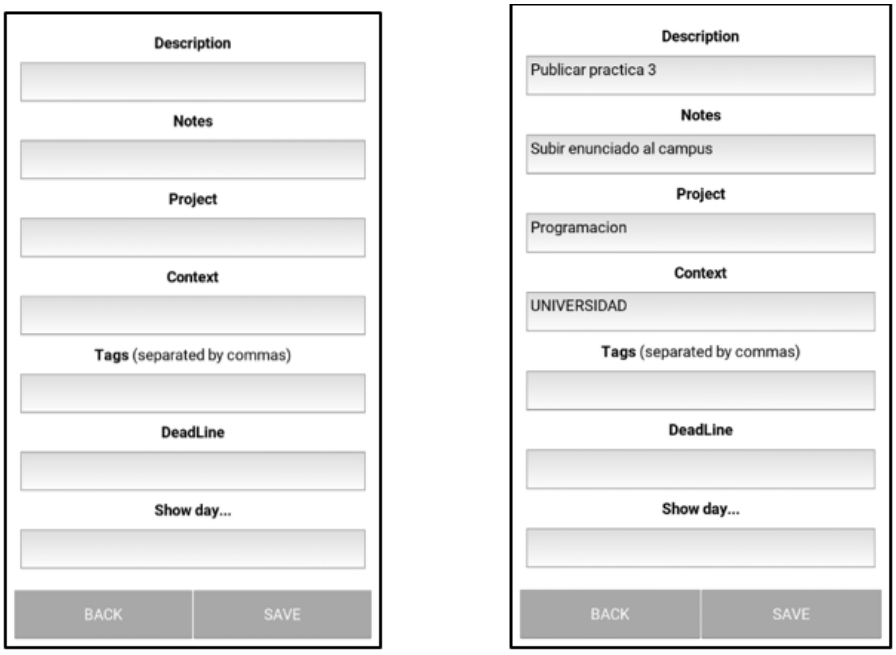

Figura 5.16: Creando una tarea

2. Editar una tarea

   Pulsando sobre el botón E(Edit) , al que podremos acceder a través de las pantallas TODAY, TICKLER, START o entrando en la información de un contexto, nos aparecerá la siguiente pantalla en la que podremos editar la tarea que deseemos. Pulsando sobre el botón *Save*, guardaremos los cambios que hayamos realizado:

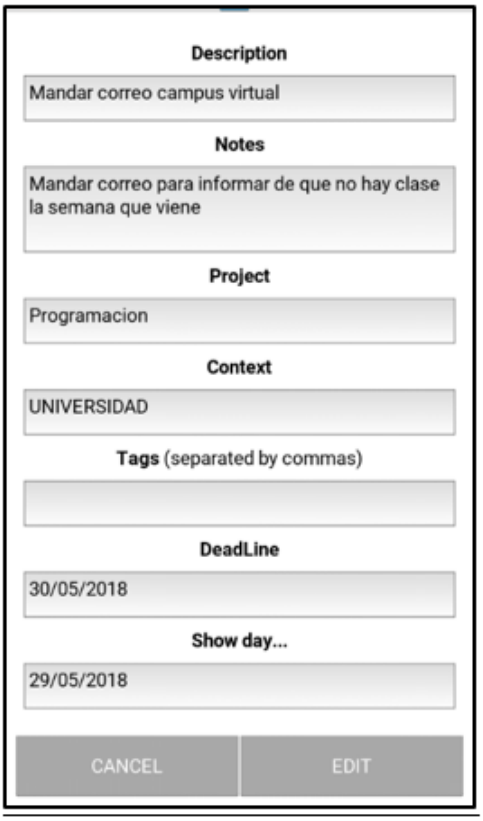

Figura 5.17: Editando una tarea

3. Eliminar una tarea

   Pulsando sobre el botón R(Remove), al que podremos acceder a través de las pantallas TODAY, TICKLER, START o entrando en la información de un contexto, podremos eliminar la tarea que deseemos.

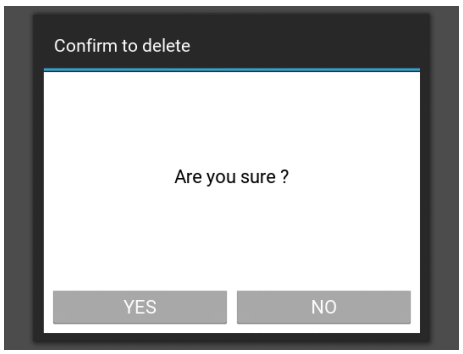

Figura 5.18: Eliminando una tarea

5.2.2. Aclaraciones adicionales sobre el funcionamiento de la aplicación

1. En las pantallas de TODAY, TICKLER y SHOW DONE, junto a cada tarea, se muestra una casilla que cambia el estado de la tarea a completada cuando se marca:

Figura 5.19: Tareas completadas

2. Es obligatorio asignar un nombre a un contexto, un nombre a un proyecto y una descripción y un contexto a una tarea. Si no lo hacemos, nos aparecerá el siguiente mensaje de error y no se guardará la acción que hayamos realizado:

Figura 5.20: Mensaje de error

3. Al crear un proyecto, si le asignamos un contexto que no ha sido creado previamente, se creará el nuevo contexto.

4. Al crear una tarea, si le asignamos un proyecto y/o un contexto que no han sido creados previamente, se crearán ambos.

5. Al eliminar un proyecto, se eliminarán todas las tareas que pertenezcan a dicho proyecto.

6. Al eliminar un contexto, se eliminarán todas las tareas que pertenezcan a dicho contexto. Por el contrario no se eliminarán los proyectos que pertenezcan a dicho contexto.

7. En el caso de que nos quedemos sin conexión a internet y realicemos un cambio sobre un contexto, tarea o proyecto existente, al recuperar la conexión nos pedirá que elijamos con qué versión nos queremos quedar. Podremos elegirlo a través de la siguiente pantalla:

Figura 5.21: Conflicto entre versiones

La opción de local se refiere al cambio que realizamos cuando no había conexión, y la opción online a la que se realizó en la aplicación web de Tracks.

Capítulo 6

Conclusiones

Hoy en día, tener una buena organización en el contexto laboral como en cualquier aspecto de nuestro día a día es uno de los aspectos más críticos. En las empresas, el uso de metodologías ágiles es una parte fundamental para maximizar la productividad. Por otra parte, desde el punto de vista de nuestras tareas diarias, necesitamos un método de organización para alcanzar el rendimiento máximo. En un mundo tan exigente no podemos dejar escapar todas las herramientas disponibles.

La metodología GTD ofrece una forma de organización rápida y eficiente, y esto unido al avance de las tecnologías y a la posibilidad de llevar en nuestros dispositivos una aplicación que aplique este método puede ayudarnos a recopilar todas las tareas pendientes y tenerlas disponibles en todo momento.

Existen muchas aplicaciones que ofrecen diferentes alternativas de organización, pero la nuestra destaca por los aspectos que, partiendo del resumen que se proporciona al inicio del documento, se han logrado construir y enumeramos a continuación:

- Es una **infraestructura multiplataforma**: conseguida tras haber desarrollado el servidor con Tracks y el cliente en código Python junto al framework Kivy.
- **Basado en Tracks**: a través del REST API proporcionado por Tracks, hemos integrado nuestra aplicación multiplataforma con la aplicación web.
- **Servidor privado**: Tras utilizar un servidor que puede ser gestionado por uno o varios usuarios, acabamos con el problema de la privacidad en la nube.

- **Modo offline**: La manera en la que está desarrollada la aplicación permite que continuemos trabajando con ella incluso cuando no tenemos conexión, lo cual resulta de gran utilidad en dispositivos móviles. Cuando el dispositivo vuelve a tener conexión a Internet, el cliente actualiza los cambios en el servidor de forma transparente para el usuario.

6.1. Valoración del TFG

Este proyecto ha sido un gran reto para nosotros, ya que se basa en tecnologías que no conocíamos o que son nuevas y existen escasa documentación sobre ellas. Concretamente, ha sido necesario familiarizarse con tres tecnologías: Tracks y el servidor, Python y Kivy.

Tracks es una aplicación con escasa documentación, y más si lo relacionamos con la instalación en una máquina y su configuración. Para los problemas que nos íbamos encontrando sólo teníamos como apoyo la página de Tracks y su repositorio de Github (ver cita [4]), por lo que las soluciones eran limitadas. Además, hemos observado que en ciertas partes la aplicación y su integración no estaban muy maduras (no funciona la conexión por token, limitación a XML).

El lenguaje que hemos utilizado para la implementación del cliente ha sido Python, ya que es el lenguaje del framework Kivy. Antes de comenzar este proyecto, sabíamos lo básico de este lenguaje, pero sin aún conocer muchas caractéristicas del lenguaje, como la orientación a objetos, hemos podido sacar el proyecto adelante.

En cuanto a Kivy se trata de una tecnología que requiere unos requisitos y condiciones muy peculiares, junto con herramientas auxiliares como Buildozer, kivy-designer y Pythonfor-android. Estas particularidades nos han hecho invertir una gran cantidad de tiempo en la búsqueda de documentación de instalación, librerías y permisos recogidos en los archivos que se precisan para lograr construir una aplicación multiplataforma robusta.

Lograr todos estos hitos nos supone una buena inyección de moral para afrontar el mundo laboral, enfrentándonos a tecnologías que apenas conozcamos y adaptándonos a ellas para lograr nuestros objetivos.

6.2. Trabajo futuro

Existen numerosas mejoras en nuestra infraestructura que podrían introducirse de cara a futuras versiones, pero que por motivos técnicos y de tiempo no ha sido posible incorporar en el software desarrollado en este TFG:

- Introducir la funcionalidad de Tracks para mostrar tareas recurrentes que se repiten en periodos de tiempo o que dependen de otras tareas para ser realizadas.
- Incluir una interfaz más sofisticada y elaborada.
- Adaptar la aplicación para que no haya que autenticarse cada vez que se inicia.
- Mejorar la pantalla de conflictos entre objetos donde se muestre más información para facilitar la elección.
- Traducir la interfaz a otros idiomas distintos del inglés.
- Crear un apartado con un calendario donde el usuario pueda consultar las tareas por fecha.
- Adaptar la compilación y la interfaz para otros sistemas operativos.

Bibliografía

[1] Allen, David. Getting Things Done. The Art of Stress-Free Productivity. Penguin Books, 2001.

[2] Phillips, Dusty. Creating Apps in Kivy. Mobile with Python. O'Reilly Media, 2014.

[3] Tracks: «Tracks, Doing Things Properly». getontracks.org. N. p., 2018. Web. `http://www.getontracks.org`

[4] TracksApp repository. getontracks.org. N. p., 2018. Web. `https://github.com/TracksApp/tracksapp.github.com`

[5] Kivy: «Open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps». kivy.org. N. p., 2018. Web. `https://kivy.org`

[6] Kivy: «Open source UI framework written in Python, running on Windows, Linux, macOS, Android and iOS». kivy.org. N. p., 2018. Web. `https://github.com/kivy/kivy`

[7] balsamiq: «Unleash Your Creativity!». balsamiq.com. N. p., 2018. Web. `https://balsamiq.com`

[8] DigitalOcean: «How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 16.04». digitalocean.com. N. p., 2017. Web. `https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-ce`

[9] SQLite: «Small. Fast. Reliable. Choose any three». sqlite.org. N. p., 2018. Web. `https://www.sqlite.org`

[10] Welcome to Python.org. Python.org. N. p., 2018. Web. `https://www.python.org/`

[11] Requests: «HTTP for Humans». Python.org. N. p., 2018. Web.
http://docs.python-requests.org

[12] Buildozer: «Generic Python packager for Android and iOS». kivy.org. N. p., 2018. Web.
https://github.com/kivy/buildozer

[13] The all new Things. culturedcode.com. N. p., 2018. Web.
https://culturedcode.com/things

[14] Chandler app. chandlerproject.org. N. p., 2018. Web.
http://chandlerproject.org

[15] Rails. rubyonrails.org. N. p., 2018. Web.
https://rubyonrails.org/

[16] Bundler. bundler.io. N. p., 2018. Web.
https://bundler.io/

[17] MySQL. oracle.com. N. p., 2018. Web.
https://www.mysql.com/

Apéndice A

Introduction

GTD stands for Getting Things Done, a term created by David Allen and explained in his book (see quote [1]). GTD is a methodology that helps a person to manage the tasks he carries out during the day, allowing him to focus on current tasks and not having to be reminded or pending of those that are to come.

GTD defines a workflow with 5 stages that make us develop our work efficiently:**Collect**, **Process**, **Organize**, **Evaluate** and **Do**. In the chapter 2.1 the concept is explained in more detail.

Next we will explain the motivation to carry out this project.

A.1. Motivation

Guaranteeing the privacy of users of GTD tools is one of the main motivations of this work. It is clear that today there are innumerable applications that exist for us to organize our day to day (see quotes [13], [14], etc.). Unfortunately, most of them have a potential problem: the collection of data that they subsequently sell to third parties.

Therefore, we propose the idea of having control over our data on a private server. Thanks to the server we can have a backup in the cloud, in addition to having our data available on multiple devices.

A.2. Project goals

As we have said before, there are innumerable applications that exist today in the market to organize your day to day but they do not guarantee that our data is safe. In addition, some do not offer offline support or the possibility of using it on all our devices. We are going to make an introduction about three characteristics of our management software, which will be expanded in the chapter 4.

A.2.1. Tracks as option

In this project we propose the use of Tracks (section 4.1.2) as a server of our GTD management software. Tracks is implemented with Ruby on Rails, it has a web server (WEBrick, although we have done it with Apache) that can be executed on any computer/server on which Ruby can be installed. Although Tracks does not have specific clients (web interface only) for different platforms, it offers a REST API that allows interaction with the server from a specifically built client application.

A.2.2. Multiplatform GTD management software

Our management software consists of two components, both multiplatform: server (Tracks), and client (kvtracks). As we have said, Tracks can be installed on any computer where Ruby is available. The client, kvtracks, is implemented in Python and, together with the Kivy framework (section 3.2.3), we get an application that can run on different operating systems such as Windows, Mac, iOS, Android, etc.

A.2.3. Offline mode

One of the most useful features in our application is the possibility of using it offline. The data is stored in a database located on the device where the client runs, and is managed automatically by the kvtracks application until the device has a connection again and the data is saved in the cloud (Tracks server). The database has a manager called SQLite (section 4.2.2).

A.3. Work plan

To achieve the objectives, the following tasks were carried out:

1. Planning of the work to be carried out by each of the members of the project team (included in the appendix D).
2. Search and read documentation about the programming languages, technologies and tools to be used.
3. Installation and deployment of the web application Tracks (server) on a virtual machine.
4. Installation of development tools (Python, Kivy, etc.) for the implementation of kvtracks.
5. Design of the sketches of kvtracks with balsamiq (appendix C)
6. Prototype development of client applications with Python to familiarize with the different technologies used.
7. Combination of previously created prototypes to check compatibility between all the technologies used.
8. Construction of the APK with Buildozer, after reading the documentation and installation.
9. Analysis of the internal operation of Tracks.
10. Modification/Adaptation of the mockups created for the exclusive use of the user interface components (widgets) supported by Kivy
11. Implementation of the two components of kvtracks: frontend and backend.
12. Integration of the two components of kvtracks.
13. Operation tests and error correction.
14. Packaging and deployment of the application.

It should be noted that the development of some of the tasks was done in parallel, and that in other tasks each team member dealt with a specific part, as agreed in task 1.

Apéndice B

Conclusions

Today, having a good organization in the workplace as in any aspect of our day to day is one of the most critical aspects. In companies, the use of agile methodologies is a fundamental part to maximize productivity. On the other hand, from the point of view of our daily tasks, we need an organizational method to achieve maximum performance. In such a demanding world we can not let all the available tools escape.

The GTD methodology offers a fast and efficient form of organization, and this, together with the advancement of technologies and the possibility of bringing an application that applies this method to our devices, can help us to collect all the pending tasks and have them available at all times.

There are many applications that offer different organizational alternatives, but ours stands out for the aspects that, starting from the summary that is provided at the beginning of the document, have been built and listed below:

- It is a **multiplatform management software**: obtained after having developed the server with Tracks and the client in Python code together with the Kivy framework.
- **Based on Tracks**: through the REST API provided by Tracks, we have integrated our multiplatform application with the web application.
- **Private server**: after using a server that can be managed by one or several users, we end up with the problem of privacy in the cloud.
- **Offline mode**: the way in which the application is developed allows us to continue working with it even when we have no connection, which is very useful in mobile

devices. When the device has an Internet connection again, the client updates the changes on the server transparently to the user.

B.1. Evaluation of the project

This project has been a great challenge for us, since it is based on technologies that we did not know or that are new and there is little documentation about them. Specifically, it has been necessary to become familiar with three technologies: Tracks and the server, Python and Kivy.

Tracks is an application with scarce documentation, and more if we relate it to the installation in a machine and its configuration. For the problems that we were finding we only had as support the page of Tracks and its repository of Github (see quote [4]), so the solutions were limited. In addition, we have observed that in certain parts the application and its integration were not very worked (the connection by token does not work, limitation to XML).

The language we have used for the implementation of the client has been Python, since it is the language of the Kivy framework. Before starting this project, we knew the basics of this language, but without knowing many characteristics of language, such as object orientation, we were able to take the project forward.

As for Kivy, it is a technology that requires very peculiar requirements and conditions, together with auxiliary tools such as Buildozer, kivy-designer and Python-for-android. These particularities have made us invest a large amount of time in the search for installation documentation, libraries and permissions collected in the files that are needed to build a robust multiplatform application.

Achieve all these milestones is a good injection of moral to face the world of work, facing technologies we barely know and adapting to them to achieve our goals.

B.2. Future work

There are numerous improvements in our GTD management software that can be introduced in future versions, but for technical and time reasons have not been possible to incorporate into the software developed in this project:

- Introduce the functionality of Tracks to show recurring tasks that are repeated in periods of time or that depend on other tasks to be performed.
- Include a more sophisticated and elaborate interface.
- Adapt the application so that it does not have to be authenticated every time it starts.
- Improve the screen of conflicts between objects where more information is displayed to facilitate the choice.
- Translate the interface into languages other tan English
- Create a section with a calendar where the user can consult the tasks by date.
- Adapt compilation and interface for other operating systems.

Apéndice C

Bocetos del diseño de la interfaz

Figura C.1: Login y pantalla principal

Figura C.2: Menú desplegable y vista de la semana

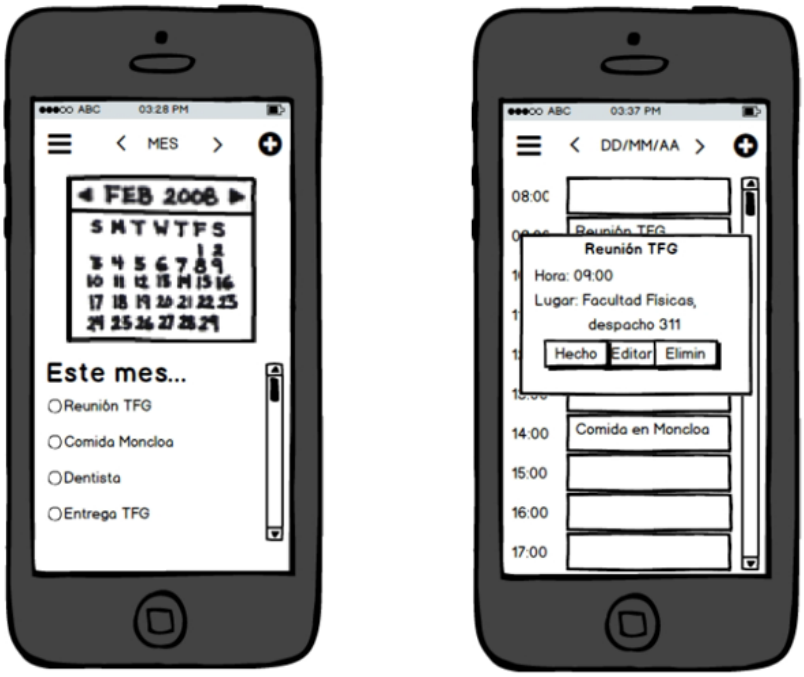

Figura C.3: Vista del mes y opciones para una tarea

Como podemos ver, la principal idea de diseño de interfaz queda muy alejada de cómo finalmente ha quedado debido a que todavía no estábamos familiarizados con Tracks, los apartados que ofrecía y la metodología GTD, pero nos sirvió para comenzar a programar con Kivy e ir viendo cómo funcionaba la creación de una interfaz (pantallas, botones, checkboxes, boxlayout, etc.).

Familiarizados con los tres principales contextos de GTD (**contextos**, **proyectos** y **tareas**), de qué manera se disponían en Tracks y estudiando distintas aplicaciones como Things 3 de iOS, comenzamos a rediseñar la aplicación manteniendo la idea de que fuera intuitiva. Para ello, decidimos que la aplicación se asemejara lo máximo posible a la interfaz web, en cuanto a la disposición de cada elemento, pero en versión móvil. Con estas ideas los resultados que obtuvimos fueron los siguientes:

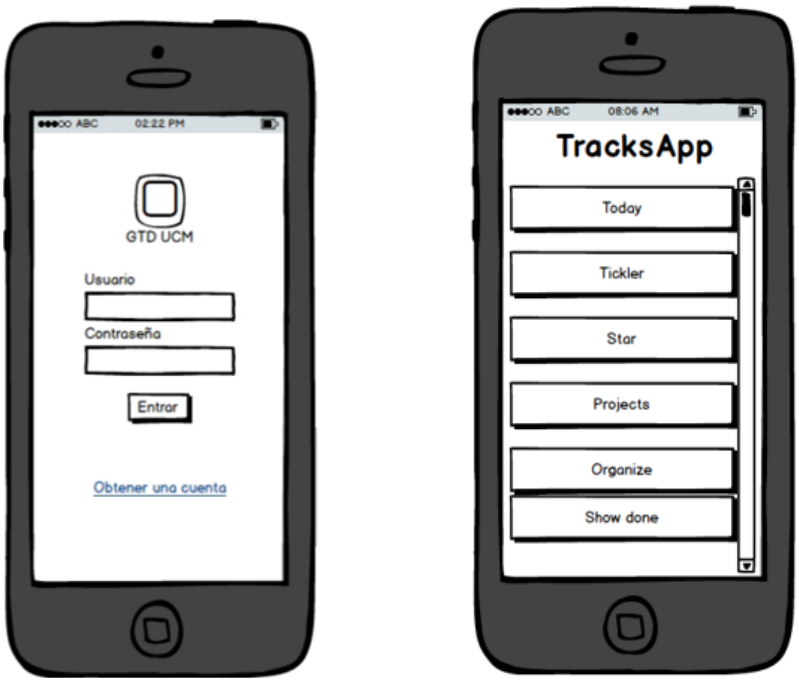

Figura C.4: Login y pantalla principal

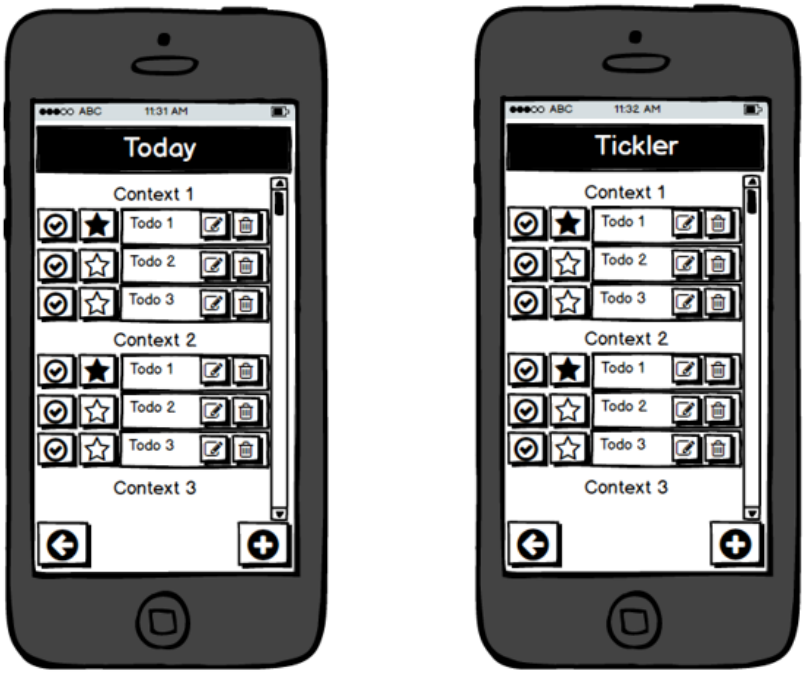

Figura C.5: Pantallas TODAY y TICKLER

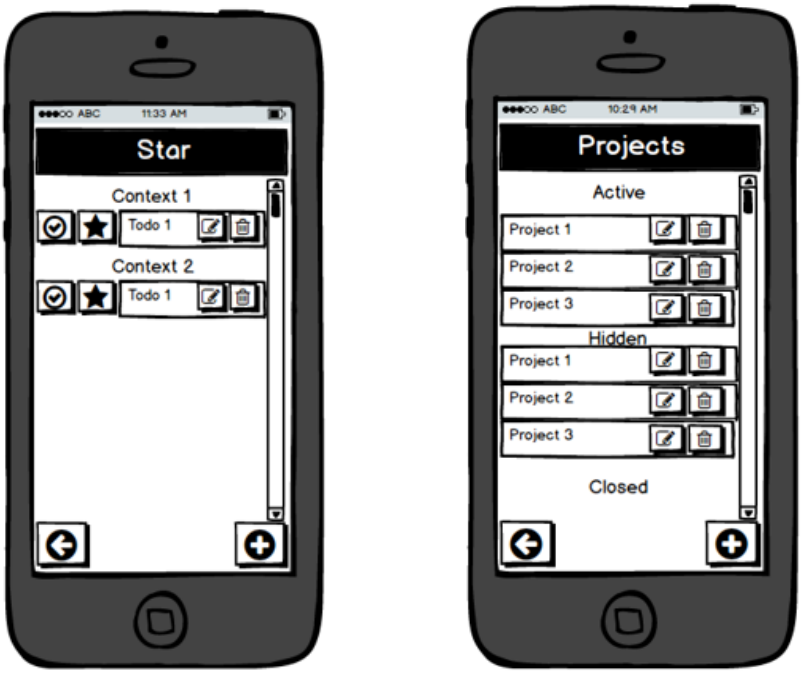

Figura C.6: Pantallas STAR y PROJECTS

Figura C.7: Pantallas ORGANIZE y SHOW DONE

Figura C.8: Vista de un proyecto

En la sección de guía de usuario (ver sección 5.2) se pueden ver bocetos de la aplicación que se ha creado.

Apéndice D

Contribución de cada miembro del equipo

D.1. Aportación de Andrés Herreros Manotas

D.1.1. Análisis y aprendizaje de la tecnología

Al inicio del curso y con ello del TFG, como se acordó en la primera reunión, se tuvo una primera toma de contacto con la metodología GTD ¿qué es? ¿Qué pretende? ¿Qué se consigue con ella? Todas estas preguntas se resolvieron buscando en internet información acerca de la metodología y del autor de la metodología. La información encontrada en internet se amplió con la lectura de libros que trataban sobre GTD.

Tras aclarar todos los conceptos que esta metodología incluye y teniendo las cosas más claras, Andrés comenzó a hacer un análisis exhaustivo sobre las tecnologías a utilizar Python, Kivy y Buildozer, que el director del TFG propuso utilizar.

Andrés tuvo libertad de investigar las tecnologías para el sistema operativo que fuera de su elección. La investigación se hizo en un principio sobre en Windows por ser el sistema operativo con el que más familiarizado estaba. Tras las adversidades con las que se encontró a la hora de instalar las distintas tecnologías, tocó volver a empezar de nuevo la investigación pero esta vez para el sistema operativo Ubuntu.

Al no disponer de un dispositivo cuyo sistema operativo fuese Ubuntu la solución por la que se optó fue utilizar una máquina virtual de VirtualBox, lo que le obligó a documentarse acerca de que versión de Ubuntu era mejor para el desarrollo con las tecnologías anteriormente citadas. Una vez que la versión fue elegida, se investigó sobre cómo montar

una máquina virtual desde cero.

Con toda la investigación realizada y todos los conceptos tanto de GTD como de las tecnologías a utilizar claros, se comenzó con el desarrollo.

D.1.2. Desarrollo

Con la información obtenida durante la fase de análisis y obviando la instalación en Windows puesto que no sirvió, lo primero que hizo Andrés fue crear una máquina virtual con el sistema operativo Ubuntu en su versión 16.04.

Configurada la máquina virtual, Andrés instaló Python y Kivy, desarrolló los primeros bocetos de la aplicación junto con Adrián para saber qué componentes tendría la aplicación (botones, títulos, cuadros de texto...) y comenzó a desarrollar pequeños prototipos de software para familiarizarse con las tecnologías. Dichos prototipos planteaban testear la creación de botones, etiquetas o cuadros de texto entre otros. Familiarizado con los principales elementos de una aplicación, Andrés comenzó a darles funcionalidad. Una vez que la funcionalidad de los elementos respondía de la manera deseada, comenzó a aplicar la información obtenida en la fase de análisis para manejar el cambio de pantallas en la aplicación.

Con este pequeño entrenamiento el siguiente paso para Andrés fue generar la aplicación para móvil, por lo que tras instalar Buildozer (ver cita [12]) en la máquina virtual, comenzó a generar los archivos APK para instalarlos en el móvil. Tras instalarlos en el móvil y comprobar su correcto funcionamiento, Andrés unió los prototipos desarrollados con los que había hecho Adrián para comprobar que todo iba bien y verificar que las tecnologías propuestas funcionaban correctamente al utilizarse conjuntamente.

Realizadas todas las pruebas posibles siendo los resultados satisfactorios, se comenzó a desarrollar la aplicación que se ha terminado entregando como resultado. Cabe destacar que durante toda la fase de desarrollo son diversos los problemas con los que nos hemos encontrado y que hemos resuelto realizando estudios adicionales.

D.1.3. Documentación

Durante las dos fases anteriormente descritas, Andrés documentó todo lo relacionado con las tecnologías analizadas y utilizadas en el proyecto.

Las distintas versiones de código se han gestionado usando Github. En la fase final de

proyecto toda la información almacenada ha sido recopilada para redactar este documento en una carpeta en Google Drive.

D.2. Aportación de Adrián Monteagudo Sampedro

D.2.1. Análisis y aprendizaje de la tecnología

Al comienzo del curso nos tuvimos documentar sobre el concepto GTD, su filosofía y metodología de organización. El profesor puso a nuestra disposición diversos libros para que comenzáramos a investigar, ofreciéndonos una idea principal para encaminarnos sobre este proyecto. Junto a esta documentación empezamos a investigar sobre Kivy y sus características. Adrián siguió sus tutoriales y construyó sus aplicaciones de prueba, que nos ofreció una guía sobre cómo funciona este framework y los requisitos que necesita. Tras esto, Adrián comenzó con el desarrollo de una aplicación de prueba.

Conjuntamente, el director del TFG nos informó de que teníamos una máquina virtual a nuestra disposición y comenzó a documentarse sobre la instalación de Tracks en un servidor externo. La información sobre esto se obtuvo consultando las páginas que ofrecía Tracks, ampliando conocimientos con otras páginas externas que incluían información sobre conexiones seguras e instalación de servidores.

En resumen, el trabajo de Adrián se centró en el backend de kvtracks y la instalación de Tracks en la máquina virtual, junto con la securización de las conexiones.

D.2.2. Desarrollo

En cuanto al desarrollo realizado por Adrián se distinguen las siguientes partes:

- Por un lado, se comenzó el desarrollo de una aplicación de prueba en Python para conocer el lenguaje. Se realizó una aplicación que verificara la funcionalidad de la REST API de Tracks mediante peticiones REST con la librería Requests de Python. Después, se diseñó la aplicación orientada a objetos con toda la funcionalidad necesaria para la ejecución de kvtracks, sus modos online/offline y la inclusión de una base de datos. Esta fue la parte a la que más tiempo dedicó Adrián.

- Por otro lado, se realizó la instalación de Tracks en la máquina Linux. Se configuró siguiendo la guía de instalación y se probó junto a la aplicación en Python. Esta

instalación se tuvo que repetir en varias ocasiones ya que la documentación no ofrece muchos detalles y se tuvo que probar varias veces hasta dar con la configuración que se adaptara a nuestras necesidades. Al ver que todo funcionaba correctamente, Adrián se dispuso a securizar las conexiones estableciendo un servidor Apache junto a la aplicación de Tracks en la máquina, y se actualizó el código Python para realizar estas conexiones seguras.

- También, Adrián aportó ayuda a Andrés realizando algunos bocetos de la interfaz de la aplicación y buscando errores que impedían seguir con la construcción del APK en Kivy. También trabajamos conjuntamente para adaptar su código del front-end con el del back-end de la aplicación, realizando las correcciones y siguiendo los consejos que nos daba el director del TFG.

D.2.3. Documentación

Para llevar un control de las versiones que se realizaban, Adrián fue almacenando en local y en Google Drive el código de la aplicación y los bocetos que realizaba. La información de Tracks se guardaba en la máquina, por lo que no requería un control de versiones específico.

Además, Adrán creó distintos documentos de Word describiendo los procesos de instalación, las tareas pendientes y las herramientas utilizadas. Estos documentos han servido de base para redactar esta memoria.